GenRad

Development Systems Division

PROPERTY OF
ElectroRent
CALL TOLL FREE (800) 423-2337
CALIFORNIA (800) 232-2173

GenRad

futuredata

MICROCOMPUTER
SYSTEMS

Z80 SELF INSTRUCTION MANUAL

for the

GENRAD 2300 ADVANCED DEVELOPMENT SYSTEM

Manual No. 2300-5025-05

5730 Buckingham Parkway
Culver City, CA 90230

Copyright 1980

2300 SERIES Z80 SELF INSTRUCTION MANUAL

The information in this document is subject to change without notice.

GenRad Development Systems Division makes no warranty of any kind with regard to this material, the contents and its purpose. GenRad D.S.D. assumes no responsibility for any errors that may appear in this document.

No part of this document may be reproduced in any form without the prior written consent of GenRad D.S.D..

WARRANTY

We warrant that this product is free from defects in material and workmanship and, when properly used, will perform in accordance with applicable GenRad specifications. If within 90 days after original shipment, it is found not to meet this standard, it will be repaired, or at the option of Genrad, replaced at no charge when returned to a GenRad service facility. Changes in the product not approved by GenRad shall void this warranty. GenRad shall not be liable for any indirect, special, or consequential damages, even if notice has been given of the possibility of such damages.

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

SERVICE FACILITIES

DOMESTIC

CALIFORNIA
GenRad Development Systems
5730 Buckingham Parkway
Culver City, CA 90230
TEL: (213) 641-7200
TWX: (910) 328-7202

SAN FRANCISCO
GenRad, Inc.
2855 Bowers Avenue
Santa Clara, CA 95051
TEL: (408) 985-0662
TWX: (910) 338-0291

ILLINOIS
GenRad, Inc.
1083 East State Parkway
Schaumburg, IL 60195
TEL: (312) 843-5580
TWX: (910) 291-1209

MARYLAND
GenRad, Inc.
1701 research Blvd.
Rockville, MD 20850
TEL: (301) 424-6224
TWX: (710) 828-9783

MASSACHUSETTS
GenRad, Inc.
300 Baker Avenue
Concord, MA 01742
(From Metropolitan Boston) TEL: 646-7409
(From all other locations) TEL: (617) 369-8770
TWX: (710) 347-1051
Cable address: GENRADCO Concord (Mass.)

FOREIGN

CANADA
GenRad, Ltd.
307 Evans Avenue
Toronto, Ontario, Canada
M8Z 1K2
TEL: (416) 252-3395
TELEX: 06967624

FRANCE
GenRad
96, Rue Orfila
75020 Paris, France
TEL: (01) 797 07 39
TELEX: 22 09 91

GERMANY
GenRad GmbH
Trausnitzstrasse 8
D-8 Munich 80, West Germany
TEL: (089) 40 18 01
TELEX: 52 99 17

UNITED KINGDOM
GenRad, Ltd.
Norreys Drive
Maidenhead, Berkshire
SL64BP, England
TEL: (628) 39181
TELEX: 84 8321

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_05483

TABLE OF CONTENTS

SUGGESTED PROCEDURE	page	1
CONVENTIONS	page	2
GETTING STARTED	page	3
FILE MANAGER	page	7
File Name Conventions	page	9
File Name Prefixes	page	11
Device Prefix	page	12
Attribute Prefix	page	13
File Attributes	page	14
Examples of Filenames	page	15
Wild Cards	page	16
Single Character Wild Card	page	17
Single, multiple or no character Wild Card	page	17
Previous Specification Wild Card	page	18
Wild Carding by Attribute	page	18
Manager Queries	page	19
Option Switches	page	20
Initialize a Diskette	page	22
Display Directory	page	25
Copy File Command	page	26
Verify File Command	page	29
Assign File Attributes	page	30
Create a File	page	31

TABLE OF CONTENTS

Scratch a File	page	32
Exchange Filenames	page	33
Rename File Command	page	34
Free Unused Space	page	35
Command Line Editing	page	36
TEXT EDITOR	page	38
Editor Display	page	40
Editor Definitions	page	41
Load Editor Workspace From a File	page	43
Write Editor Workspace to a File	page	44
Write Editor Workspace and then Load	page	45
End Output File	page	46
Advance Edit Line	page	47
Backup Edit Line	page	48
Delete Lines	page	49
Clear Edit Workspace	page	50
Set and Clear Tabs	page	52
Set Upper Case Mode	page	57
Set Upper/Lower Case Mode	page	57
Set Upper/Lower Case Toggle	page	58
Line Insert Mode	page	59
Line Editing Mode	page	61
Character Inset Sub Mode	page	62
Special Key Board Editing Operation	page	63

TABLE OF CONTENTS

Find and Replace String in Workspace page 67

Get and Replace String in File page 70

Block Move Commands page 71

Place Marker Command page 72

Copy Command page 73

Copy From Command page 74

Copy to Command page 75

Kill Command page 76

Move Command page 77

ASSEMBLER page 79

Assembler Options page 81

Source File File page 83

Macro File page 84

Output File page 85

Listing File page 86

Restarting the Assembler page 89

Examples of Assembler page 90

LINKER page 92

Linker Options page 94

Specify Input File page 96

Specify Output File page 97

Listing File page 98

Linker Input page 100

#ORG page 101

TABLE OF CONTENTS

#END	page	102
Examples of Linker	page	103
DEBUGGER	page	114
Debugger Display	page	116
Disassembly Display	page	118
Memory Map Command	page	119
Memory Pointer	page	120
Command Entry Cursor	page	120
Load Program	page	121
Swap Command Cursor	page	124
Display Memory	page	125
Position Memory Cursor	page	126
Store Data in Memory	page	128
Store Data in Register	page	129
Find Data in Memory	page	131
Breakpoints	page	132
Debugger Hardware Breakpoint	page	133
Software Breakpoints	page	133
Breakpoint Set Command	page	134
Breakpoint Display Command	page	134
Breakpoint Reset Command	page	135
Breakpoint Clear Command	page	135
Execute Single Instruction	page	136
Trace Execution	page	137

TABLE OF CONTENTS

Execute Program	page	139
No-Interrupt Execution Command	page	140
Write Protect Memory	page	141
Write Program to File	page	142
EPROM Programming	page	143
Check for EPROM Empty	page	144
Program EPROM	page	144
Input Data From EPROM	page	145
Verify Data in EPROM	page	145
Z80 IN-CIRCUIT EMULATOR	page	146
Z80 Emulator Features	page	148
Emulator Mode Command	page	150
Users Memory Map Command	page	154
LOGIC ANALYZER	page	156
Logic Analyzer Entry	page	158
Logic Analyzer Exit	page	159
Logic Analyzer Display	page	160
And/Or Provision	page	161
Logic Analyzer Breakpoints	page	162
Breakpoint Parameters	page	163
Logic Analyzer Trace Qualifier	page	167
Trace Qualifier Parameters	page	168
Logic Analyzer Qualifier Register Examples	page	171
Logic Analyzer Timer	page	172

TABLE OF CONTENTS

Default Contitions	page	173
Logic Analyzer Display	page	173
Parallel State Display	page	175
Serial (Timing Diagram) Display	page	177
Executing a Program with Logic Analysis	page	180
Execution from the Logic Analyzer	page	181
Execution From the Debugger	page	182

SUGGESTED PROCEDURE

The suggested procedure is to scan this manual once as an overview without using the self-instruction diskettes. Each section of the manual has a title page with the module name and a brief summary of features associated with that module.

After completing the overview, go through the manual a second time using the diskettes to note examples of features of each module.

Only two keys are used to go through the Z80 Self Instruction modules. When a command line is blank (the cursor is next to a ">"), enter a "^" to continue with the module. When the command line contains a command, enter <RETURN> to accept the command.

Each sample display is separated from the text by a double row of asterisks. If there are two sample displays on a page, they are separated with a triple row of asterisks.

NOTE:

If you should encounter a SYNTAX ERROR in any of the modules or any other error that will not let you proceed, you must restart the module by typing "J^Sn" "n" being the number of the module you wish to use.

CONVENTIONS

LETTERS:

Upper case — Commands are printed in upper case and must be typed exactly as shown.

Lower case — Descriptive terms are all in lower case and should not be typed as shown, but substituted with the needed value. All parameters will appear in lower case.

PARAMETERS:

[ ] — Brackets are used to enclose optional parameters.
[parameter]

{ } — Braces are used to enclose all required parameters.
{parameter}

Brackets and braces are shown for clarification purposes only; they are not part of the input.

DOTS:

. . . — A series of horizontal or verical dots in command syntax indicates the parameter preceding the dots may be repeated indifinitely.

KEYS:

< > — If a special character key is required, it will be enclosed by < and > (eg. <RETURN> is return key.

ASTERISKS:

********** — A series of asterisks in two rows will mark the top and bottom of a screen display. Three rows of asterisks are put between two screens on one page.

GETTING STARTED

1. After unpacking the 2300 ADS console and the disk drive unit from the two boxes, place them next to each other on a table.

2. Remove the back of the console and make sure all cards are in their guides and firmly plugged into the bus.

3. Plug the disk drive power cord into the socket on the back of the console (next to the on/off switch.)

4. Take the flat cable extending from the back of the disk drive and plug it into the cable extending from the back of the console.

5. Plug the console into an electrical outlet.

6. Turn the power switch on the back of the 2300 system to "ON".

7. Turn the power switch on the front of the disk drives to "ON".

8. When the 2300 system is powered on, two lamps should be lit on the front right of the display panel. The top light is labelled "ON" and the light underneath is labelled "RUN". The top line of the CRT should have the characters BOOT80D on the left and the futuredata logo on the right. The second line should have a ">" in the first column and a flashing rectangle next to it. This is the boot message.

   If the boot message does not appear, press the LOAD key in the upper right corner of th keyboard. If the message still does not appear, try adjusting the brightness of the CRT. The brightness control knob is under the console on the right side. If there is no message after turning the brightness control knob to maximum intensity, make sure that the cards are seated properly in the card cage and that the load key was depressed. If there is still a problem, call the local GENRAD service office or an applications engineer at GENRAD/ Development Systems Division in Los Angeles. DO NOT PROCEED FURTHER UNTIL THE SELF INSTRUCTION MANUAL IS READ THOROUGHLY.

9. When the boot message appears, place the disk labelled "Z80 Self Instruction Disk Drive 0" in Drive 0 with the disk's label to the right. When the disk catches and doesn't pop out, close the disk drive door by moving it to the left until it locks.

10. Place the disk labelled "Z80 Self Instruction Disk Drive 1" in Drive 1 and close the disk drive door.

11. Type the keys J and M on the keyboard consecutively. This causes JM to appear on the screen between the ">" and the cursor. Enter a <RETURN> and the system will jump to the File Manager.

12. We will now look at the directories of the disks before starting the Self Instruction module.

13. Enter a D followed by a <RETURN>. This will display the directory of the disk in Drive 0. The display is shown below

Directory of Drive 0 Disk

```
********************************************************************************
********************************************************************************
======== MANAGER V02 ==========================(futuredata)========
>

FILES ON DRIVE 0:          7 FREE TRACKS

   PW  1 DIR            PR  1 INCR.R        PWS  1 TEXT0
       1 00-5804-05    PWS  1 INSERT       PWS  4 TEXT1
    Z  5 ASMRZ80       PWS  1 INSERTMODE   PWS  4 TEXT2
  PWO  1 BLANK          Z  3 LINKER80      PWS  2 TEXT3
  PWO  1 COUNTER        Z  4 MANAGER80     PWS  2 TEXT4
  PWS  1 COUNTER.N     PWS  1 S0           PWS  4 TEXT5
  PWR  1 COUNTER.R     PWS  1 S1           PWS  2 TEXT6
    Z  3 DEBUGZ80L     PWS  1 S2           PWS  4 TEXT7
  PWS  1 DELETE        PWS  1 S3           PWS  1 TIMER.N
       1 DUM1          PWS  1 S4           PWR  1 TIMER.R
       1 DUM2          PWS  1 S5           PWS  1 TREE
    Z  3 EDITOR80      PWS  1 S6             Z  1 ^CFP80
  PWS  1 ENDEDIT       PWS  1 S7
  PWO  1 INCR           PS  1 SAVE
  PWS  1 INCR.N        PWS  1 SCROLL

END OF LIST
********************************************************************************
********************************************************************************
```

13. Next enter D1 followed by a return. This will display the directory of the disk in Drive 1. It should be the same as the display below.

Directory of Drive 1 Disk

```
**********************************************************************************
**********************************************************************************
======== MANAGER V02 ==========================================(futuredata)=======
>


FILES ON DRIVE 1:        57 FREE TRACKS

    PW  1 DIR
     Z  5 ASMRZ80
     Z  3 DEBUGZ80L
     Z  3 EDITOR80
     Z  3 LINKER80
     Z  4 MANAGER80
     Z  1 ^CFP80



END OF LIST
**********************************************************************************
**********************************************************************************
```

15. Now enter J^SO followed by <RETURN> to get started with the Self Instruction module.

Z80 SELF INSTRUCTION DISK FOR THE GENRAD 2300 ADS

TO PROCEED, PLEASE TYPE THE NAME OF THE FEATURE YOU DESIRE TO SEE

NAME	FEATURE DEMONSTRATED
S0	CURRENT PROGRAM
S1	FILE MANAGER
S2	EDITOR
S3	ASSEMBLER
S4	LINKER
S5	DEBUGGER
S6	IN-CIRCUIT EMULATOR
S7	LOGIC ANALYZER

To start a module, type S, the number of the module and then <RETURN>. For instance, to start the File Manager module, type S1<RETURN>.

After each of the modules, control returns to the user to select another module. If you wish to exit the Self Instruction modules, depress <LOAD> and the boot message will be displayed

GenRad/DSD

Advanced Development System

FILE MANAGER

TO CONTINUE TYPE ^

Features:

Initialize diskettes
Create, Rename, Copy files
Display the Directory
Print files
Initialize serial ports

To continue, enter ^.

FILE MANAGER

JM

This command invokes the Manager.

TO CONTINUE TYPE ^

After entering ^, the module will enter the command "JM" and jump to the Manager. The top line of the CRT displays the GenRad DSD logo and the heading "MANAGER V02". The second line of the display is the user command line. The first character on the second line is ">". The blinking rectangle next to ">" is the cursor. All commands to the Manager are entered here.

To continue, enter ^.

FILE NAME CONVENTIONS

[new/old spec:][device spec:][(attribute spec)]{wild-filename}

The above is the most general filename specification for the File Manager.

TO CONTINUE TYPE ^

[N:][device spec:]{filename}

This is the filename specification for ADS operating system programs other than the File Manager.

The nomenclature is explained in the next several displays.

Subsequent displays will describe the syntax of a filename in more detail

The term wild-filename is described on page 16 of this manual.

To continue, enter ^.

FILENAME PREFIXES

There are three types of specifications which may precede the filename itself. These are:

TO CONTINUE TYPE ^

NEW or OLD PREFIX	[new or old spec:]
DEVICE PREFIX	[device spec:]
ATTRIBUTE PREFIX	[(attribute spec)]

>

NEW/OLD PREFIX

[new or old spec:]

If a file specified in a Manager command does not exist, the Manager will automatically create a new file. If one already exists, it will be used. The New/Old prefix allows the user to override this default.

TO CONTINUE TYPE ^

N:

This instructs the Manager to creat a new file. If the file file already exists, the message OLD FILE EXISTS is displayed.

O:

This instructs the Manager to use only files which exist on diskette. If the file does not exist, the message OLD FILE NOT FOUND is displayed.

>

DEVICE PREFIX

The device prefix indicates the physical source or destination of the file. Disk drives, the printer and serial output ports are treated as file oriented devices.

TO CONTINUE TYPE ^

---

n:

The user may specify the desired disk drive for the specified file: 0<=n<=3 for a 2300 development system; 0<=n<=9 for a 2301 development system network station.

DEVICE PREFIX

P:

When P: is specified as the output file, the specified file is sent directly to the Centronics compatible parallel printer port.

TO CONTINUE TYPE ^

P may be specified as the destination file in the Copy File command in the Manager. This allows the user to print a source file on a Centronics compatible printer.

P: may be specified as the listing file in the Assembler or in the Linker. It may also be the output file in the Editor.

DEVICE PREFIX

S{n}:

When S{n}: is specified as the output file, the specified file is sent directly to serial port n. n defaults to the value specified in the Initialize serial port command in the Manager.

TO CONTINUE TYPE ^

S{n}: may be specified as the destination file in the Copy File command in the Manager. This allows the user to print a source file to the desired serial port.

S{n}: may be specified as the listing file in the Assembler or in the Linker.

>

ATTRIBUTE PREFIX

[(attribute spec)]

The Attribute prefix is used to specify the file attributes on an output file in a Manager command. It may also be used in conjuction with the wild card filename features to specify a group of files with common attributes.

TO CONTINUE TYPE ^

To use the attribute spec, place the desired attribute(s) in parentheses after the device spec and just before the filename.

(R)YEEN.R

Specifies the relocatable file named YEEN.R .

>

FILE ATTRIBUTES

File attributes describe the nature of the information and the level of protection of the information in the file. Attributes also are used to differentiate between system files and user files.

TO CONTINUE TYPE ^

Attribute	Description
S	Source (text) file
R	Relocatable object file (assembler output)
O	Absolute object file (linker output)
P	Permanent file (can not be deletes from disk)
W	Write protected file
Z	Operating system file
C	Command file (sequence of system commands)

File attributes are used by operating system programs to verify that a file is being used properly. For instance, a file must have an S attribute (be a source file) to be loaded into the Editor.

Internally GENRAD/DSD uses a filename convention which supplements the file attributes. If ".N" is appended to a filename, it is the newest copy of the source code; ".O" is appended to a backup copy or old copy of the source; ".R" is appended to a relocatable object file; ".C" is appended to a command file; ".D" is appended to a document or description file; and no characters are appended to the absolute object code file.

For example:

TEST.N	Newest source file.
TEST.O	Backup copy of source file.
TEST.R	Relocatable object file.
TEST	Absolute object file.
TEST.D	Documentation file.
TEST.C	Command file.

To continue, enter ^.

EXAMPLES OF FILENAMES

1:(S)YEENETTE	Source file on Drive 1 called YEENETTE
N:DOUGHBOY	New file on Drive 0 called DOUGHBOY
S2:	Serial port 2

TO CONTINUE TYPE ^

P:	Printer port
(PWO)"4BLANK"	Permanent, write protected, absolute object file on Drive 0 called 4BLANK
0:3:(SP)"DUM""DUM"	An existing, permanent source file on Drive 3 called DUM"DUM

>

All of the preceding examples of filenames refer to only one file. At times it is faster to specify a group of files having common attributes, a common sequence of characters in the filename or a combination of the two.

The next several displays will explain how to specify files in groups rather than singly.

To continue, enter ^.

****

WILD CARDS

Manager commands will not only work on single files or pairs of files, but will also work on groups of files which have some part of their name in common. For example, instead of copying YEEN, YEEN.N, YEEN.O, and YEEN.R with four separate Copy File commands, they may all be moved in one command through the use of wild cards.

TO CONTINUE TYPE ^

There are three separate wild cards which are explained below. Files may also be grouped by attributes.

Wild-spec in a Manager command stands for a filename which may include one or more of these wild cards.

****

Wild cards are a very powerful feature of the Manager; therefore, care must be exercised in using wild cards to avoid unexpected and undesirable results.

Wild cards can represent any alphanumeric character or a period. The entire filename must be surrounded by quotation marks, if a wild card is to represent any other character or a character other than a letter for the first character in that file name.

To continue, enter ^

SINGLE CHARACTER WILD CARD

?

The question mark character represents any single alphanumeric character or period occurring in the same position in the filename. The question mark may appear more than once in a filename.

TO CONTINUE TYPE ^

YEEN.? would specify the group of files which start YEEN. and have any last character. This would include YEEN.N, YEEN.O, and YEEN.R but not YEEN. or YEEN.LC since they have none or two characters and not one.

SINGLE, MULTIPLE OR NO CHARACTER WILD CARD

*

The asterisk character represents any zero to ten characters occurring in the same position as the asterisk in the filename. The asterisk may appear more than once in a filename.

TO CONTINUE TYPE ^

YEEN* would specify the group of files which start with YEEN. This would include YEEN.N, YEEN.O, YEEN.R, YEEN1, YEEN and YEEN.LCF. This would not include YEN .

PREVIOUS SPECIFICATION WILD CARD

$

The dollar sign may be used as the destination filename in a Manager command to specify the same name as the source filename. This wild card is most often used in the Copy File command.

TO CONTINUE TYPE ^

This wild card allows the user to enter shorter commands to the Manager. For example, MYEEN*,1:YEEN* is equivalent to MYEEN*,1:$.

WILD CARDING BY ATTRIBUTE

Files may be specified in group according to common filenames by using the wild cards. Files may also be grouped by common attributes by combining an attribute prefix and a wild-spec.

TO CONTINUE TYPE ^

1:(S)*.N	All source files on Drive 1 which start with a letter and end with .N
(Z)"*"	All system files on Drive 0 including files whose names do not start with a letter or contain characters other than alphanumeric and a period.

MANAGER QUERIES

Verification is requested for many of the Manager commands before executing the command. There are several possible responses to these queries.

Response	Function
Y	Process the command.

TO CONTINUE TYPE ^

<BREAK>	Abort the command. If some action is underway, the action is aborted and the next operation started.
N <STEP> <TAB> <RETURN>	These responses cause this command or this instance of the command (when using wild cards) to be skipped. Exceptions will be noted.

To continue, enter ^.

OPTION SWITCHES

Option switches may be appended to Manager commands to print messages on the printer, suppress queries, suppress the end of screen pause and to force queries.

These switches are commonly used in conjunction with commands involving wild cards.

TO CONTINUE TYPE ^

/A

This switch suppresses the query given to some commands prior to execution. All files will be processed as if a Y were entered in response to each query.

Caution should be excercised when using this switch.

/C

This switch has the same effect as the /A switch except it also suppresses the end of screen pause. When using wild cards in a Manager command, the screen may fill up with messages. When the screen is full of messages the user must type a key (any key) to clear the screen and continue. The /C switch suppresses this pause.

TO CONTINUE TYPE ^

/Q

This switch forces the Manager to query the user for verification before any action is taken.

/P

The /P switch sends all messages displayed on the screen to the Centronics compatible parallel printer port. This information may be sent to a serial printer by modifying the printer driver on the Utility disk and configuring a new Manager.

TO CONTINUE TYPE ^

/V

The /V switch will automatically compare the source and destination files with the Copy File (M) command. This switch is only used with the Copy File command.

To continue, enter ^.

INITIALIZE A DISKETTE

I[drive-number][/S=sector spacing][/C][/V]

Initializing a diskette defines the sector boundaries and establishes the directory. Any information on the disk prior to initialization is lost in this process. All factory fresh disks must be initialized before they can be used by the system.

EXAMPLE NEXT, T CONTINUE TYPE ^

---

drive-number is the disk drive containing the disk to be initialized.

sector spacing defines the number of physical sectors separating logical sectors on the disk. 4<=sector-spacing<=12.

The optimal spacing for most Z80 ADS system programs is 5; the default spacing is 8.

The diskettes used by the system should be IBM-compatible, soft-sectored, single-sided, double-density. Each the diskette has a small notch on one side which is the write protect notch. This notch must be covered with some non-transparent material to enable writing on the disk. The disk is inserted into the drive with the label facing right.

If the user forgets to cover the write protect hole and tries to initialize the disk, the manager will issue the message: WRITE PROTECT.

Optimal spacing means the best spacing to perform reads and writes at maximum speeds.

When ^ is entered, the module jumps to the Manager and displays the command I1/S=5.

To continue enter <RETURN>. The module enters a Y in response to the request for verification, and the disk in Drive 1 will be initialized with a sector spacing of 5. When the Manager has finished initialization (about 45 seconds later), the directory will be displayed and the cursor will return to the command line.

To continue, enter ^.

INITIALIZE SERIAL PORT

IS[n:][/BAUD=300][/PARITY=letter][LENGTH=7][/STOP=1]

This command allows the user to change the default parameters associated with the serial port.

n is the serial port number. n=1 or 2.

TO CONTINUE TYPE ^

BAUD is the baud rate for transmission and defaults to 300. The legal BAUD rates are:

50	150	1800	7200
75	300	2400	9600
110	600	3600	19200
134.5	1200	4800	

>

INITIALIZE SERIAL PORT

IS[n:][/BAUD=300][/PARITY=letter][LENGTH=7][/STOP=1]

PARITY defaults to NONE. Legal values for PARITY are:

NONE EVEN ODD

EXAMPLE NEXT, TO CONTINUE TYPE ^

LENGTH is the length of a character in bits. It defaults to seven bits. 5<=LENGTH<=8.

STOP is the number of stop bits. It defaults to one stop bit. Legal values for STOP are:

1 1.5 2

>

SERIAL PORT INITIALIZATION

Once the serial ports are initialized by the Manager, they need not be initialized when using other system programs. The user may change the parameters when necessary.

When ^ is entered, the module jumps to the Manager and displays the command IS1:/BAUD=4800.

When <RETURN> is entered, the baud rate for serial port 1 is changed from 300 to 4800.

When ^ is entered, the command

IS2:/STOP=1.5/BAUD=300/PARITY=ODD

is displayed.

When <RETURN> is entered, the parameters for serial port 2 will be changed. The user is now free to experiment with the Initialize Serial Port command

When <RESET> is entered, all serial port values are set to their defaults.

To continue, enter ^.

DISPLAY DIRECTORY

D[drive-number][/C][/P]

D{wild-spec}[/C][/I]

EXAMPLE NEXT, TO CONTINUE TYPE ^

Displays the directory of the specified disk. The directory files are displayed in alphabetical order in one to three columns. The attributes associated with each file and the number of tracks allocated to each file are displayed preceding to each filename. Drive-number defaults to 0. If a wild-spec is entered, only files matching the wild-spec will be listed.

When ^ is entered, the module jumps to the Manager and displays the D command.

When <RETURN> is entered, the directory of the disk in Drive 0 is displayed and the command D1 is displayed.

When <RETURN> is entered, the directory of the disk in Drive 1 is displayed and the command D(S)* is displayed.

When <RETURN> is entered, all source files on Drive 0 are displayed and the command DTEXT? is displayed.

When <RETURN> is entered, all files on Drive 0 which are five characters long and start with TEXT are displayed.

The user is now free to experiment with the Display Directory command.

To continue, enter ^.

****

COPY FILE COMMAND

```
M{from file-spec},{to file-spec}[/V]

                                        [/A]
M{input wild-spec}[,output wild-spec][/C][/P][/V]
                                        [/Q]
```

TO CONTINUE TYPE ^

```
                                      [/A]
M{input wild-spec},{file-spec}[/C][/P][/V]
                                      [/Q]
```

The Copy File command allows the user to copy files to another disk file, to a serial port or to the parallel printer port. The copied files remain unchanged in the "from" location.

****

COPY FILE COMMAND

```
M(Z)*,1:"$"/V
```

This command will copy all system files from Drive 0 to Drive 1 and verify that the files have been copied correctly

---

TO CONTINUE TYPE ^

```
M*,1:"$"/C/V
```

This command will copy all files from Drive 0 to Drive 1, verify that the copy was correct and suppress the end of screen pause.

****

COPY FILE COMMAND

M(S)*,1:(SPW)"$"/V

This command will copy all source files from Drive 0 to Drive 1, verify that the files have been copied correctly and give the files on Drive 1 the source permanent and write protect attribute

EXAMPLE NEXT, TO CONTINUE TYPE ^

MYEEN.N,P:

This command will print YEEN.N on a Centronics compatible printer. YEEN.N may be a source file, a relocatable object file, or an absolute object file.

The following commands may also be used to copy a diskette in Drive 0 to a diskette in Drive 1:

All system files on Drive 0 copied to Drive 1	X0,1,S
All files on Drive 0 copied to Drive 1	X0,1,A
All source files on Drive 0 copied to Drive 1	X0,1
All source files on Drive 1 copied to Drive 0	X1,0

When ^ is entered, the module will jump to the Manager and display the command MTEXTO,DUM1

When <RETURN> is entered, the information in file TEXTO is copied over the information in file DUM1. DUM1 is also given the attributes of TEXTO.

When ^ is entered, the command MDUM1,1:$ is displayed.

When <RETURN> is entered, the file DUM1 is copied to a file on Drive 1 with the same name. Note that the file is automatically created since it did not previously exist on Drive 1. The directory of the disk in Drive 1 is displayed.

When ^ is entered, the command M(Z)*,1:$ is displayed.

When <RETURN> is entered, all files with a Z attribute on Drive 0 are copied to Drive 1.

When ^ is entered, the command MS?,1:$/Q is displayed.

When <RETURN> is entered the Manager asks if file SO should be copied to Drive 1. The Manager prompts for all files that are two characters long and start with S. The user indicates which files are to be moved by typing Y or N.

The user may now experiment with the Copy File Command.

To continue, enter ^.

VERIFY FILE COMMAND

V{file-spec}[,file-spec]

V{wild-spec}[,wild-spec]

This command compares the contents of the two specified files to be sure they are identical.

EXAMPLE NEXT, TO CONTINUE TYPE ^

This command does not alter the information in the files being compared.

>

When ^ is entered, the module will jump to the Manager, display the directory and display the command VEDITOR80,1:$.

When <RETURN> is entered, the Manager will compare EDITOR80 on Drive 0 with EDITOR80 on Drive 1. The files should be identical and thus should verify. The manager will display FILE VERIFIED when the comparison is completed successfully.

When ^ is entered, the command VTEXT1,TEXT2 is displayed.

When <RETURN> is entered, the two files are compared. Since they are not the same the Manager displays the message BAD VERIFY.

The user may now experiment with the Verify File command.

To continue, enter ^.

ASSIGN FILE ATTRIBUTES

```
A{file-spec}[/P][A=attribute(s)]

          [/A]
A{file-spec}[/P][/A=attribute]
          [/Q]
```

EXAMPLE NEXT, TO CONTINUE TYPE ^

This command allows the user to change the attributes of a file. File attributes help prevent the misuse or accidental destruction of files.

Data Type Attributes
- S - Source file
- R - Relocatable object file
- O - Absolute object file
- Z - Operating system file

File Protection Attributes
- P - Permanent file
- W - Write protected file

Other Attributes
- C - Command file

When ^ is entered, the module jumps to the Manager, displays the directory and the command "ADUM1/A=S".

When <RETURN> is entered, "DUM1" is given the "S" attribute.

When ^ is entered, the command "ADUM1/A=ROS" is displayed

When <RETURN> is entered, DUM1 is given the "ROS" attributes. This combination of attributes is not a meaningful combination.

When ^ is entered, the command "ADUM1/A=" is displayed.

When <RETURN> is entered, the attributes are deleted from the file DUM1.

When ^ is entered, the command "A1:S?" is displayed.

When <RETURN> is entered, the Manager allows the user to selectively change the attributes of the files on Drive 1 which start with an "S" and are two characters long.

The user is now free to experiment with the Assign Attributes command. The files on Drive 0 should all be restored to their original attributes if they are altered before continuing.

To continue, enter ^.

CREATE A FILE

C{file-spec}[/I=initial-alloc][/E=extension-alloc]][/C][/P]

The Create File command allows the user to create a file in the Manager.

EXAMPLE NEXT, TO CONTINUE TYPE ^

Initial-alloc is an integer specifying the number of tracks to be allocated to the newly created file.

Extension-alloc is an integer specifying the number of tracks to be added to the total track allocation when file overflows. The former occurs during a write operation.

>

Files may be created in any portion of the operating system by specifying "N:" before an output filename.

When ^ is entered, the module jumps to the Manager, displays the directory of Drive 0 and displays the command CDUMMY/I=2.

When <RETURN> is entered, a file named DUMMY will be created on Drive 0. The file is two tracks long and has no attributes. The system will allocate the two free tracks closest to the outside of the disk. These tracks do NOT have to be contiguous. The command C1:(S)DUMMY is displayed.

When <RETURN> is entered, a file named DUMMY will be created on Drive 1 and given the "S" attribute.

The user may now experiment with the Create File command.

To continue, enter ^.

SCRATCH A FILE

S{file-spec}

[/A]
S{wild-spec}[/C][/P]
[/Q]

EXAMPLE NEXT, TO CONTINUE TYPE ^

This command deletes the specified file from the directory and releases the tracks which had been allocated to that file.

The Manager will request verification before scratching the file, allows the user to verify that the file is to be deleted. The user should enter a "Y" will delete the file. Any other input will leave the file unchanged. Files with P or Z attribute cannot be scratched.

When ^ is entered, the module jumps to the Manager and creates a file named DUMMY on Drive 0 that is 1 track long. The command SDUMMY is also displayed

When <RETURN> is entered, the .anager deletes DUMMY after verifying that the file should be deleted. After the file is deleted, the directory of the disk from which the file was deleted is displayed on the screen.
(Note: The "Y" was supplied in response to the Manager query so that the module could have a standard response.)

If a permanent (P) or system (Z) file is deleted, the message PERM FILE is displayed and the Scratch file command is ignored.

When ^ is entered, the command S1:S? is displayed.

When a <RETURN> is entered, the Manager will prompt the user so that the files on Drive 1 which start with an S and are two characters long may be selectively deleted.

The user may now experiment with the Scratch File command.

To continue, enter ^.

CAUTION: THERE IS NO WAY TO RECOVER INFORMATION FROM A FILE THAT HAS BEEN SCRATCHED FROM THE DISK.

EXCHANGE FILENAMES

E{file-spec},{file-spece}

E{wild-spec}[,wild-spec][/A][/C][/P][/Q]

EXAMPLE NEXT, TO CONTINUE TYPE ^

This command exchanges the names of two files or two groups of files. All other information is left unchanged. This command is very useful in saving a second copy of a file and retaining a set of naming conventions.

When ^ is entered, the module jumps to the Manager, displays the directory of Drive 0 and displays the command EDUM1,DUM2.

When <RETURN> is entered, the filenames will be switched. This will not alter the information in either of the files and since the files are 1 track long, the directory does not appear to change.

When ^ is entered, the module displays the command EDUM1,DUM2.

When <RETURN> is entered, the files will be restored to their original names.

To continue, enter ^.

FREE UNUSED SPACE

F{file-spec}

[/A]
F{wild-spec}[/C][/P]
[/Q]

EXAMPLE NEXT, TO CONTINUE TYPE ^

This command releases unused diskette tracks from a file.

If new file information is written over existing information in a file and the new file information does not require as much space, the unused diskette tracks may be returned to system availability by using this command.

When ^ is entered, the module jumps to the Manager, creates the file DUMMY, allocates two tracks and displays the command FDUMMY.

When <RETURN> is entered, one of the two tracks allocated to DUMMY is released to the system since it contains no information.

When ^ is entered, the command F1:(S)"*"/C is displayed.

When <RETURN> is entered, all empty tracks in source files on Drive 1 will be released to the system.

The file DUMMY is then deleted for the next time this module is executed.

To continue, enter ^.

COMMAND LINE EDITING

<CTRL>-<O>

When these two keys are depressed simultaneously, the last command entered to the Manager will be recalled to the command line. This allows the user to correct an error in a command or to issue a series of similiar commands with a minimum of typing.

TO CONTINUE TYPE ^

<BACESPACE>

This key moves the cursor back one space on the command line and deletes the character under the cursor.

COMMAND LINE EDITING

<CAN>

This key deletes the entire command line from the display when it is depressed.

TO CONTINUE TYPE ^

<DEL>

This key deletes the character under the cursor and moves all of the characters to the right of the cursor one space to the left.

COMMAND LINE EDITING

<INS>

This key shifts to Insert mode and an asterisk appears below the cursor. In the Insert mode the characters to the right of the cursor are moved one space to the right as each new character is entered.

TO CONTINUE TYPE ^

<-- -->

The left and right arrows allow the user to position the cursor on the command line without destroying the characters already there.

NOTE: The cursor must be to the right of all the characters to be entered as a command.

>

FILE MANAGER

This concludes the Manager module.

TO CONTINUE TYPE ^

The user will be returned to the menu to make the next selection.

>

To continue, enter ^, and then select module 82.

GenRad/DSD

Advanced Development System

CRT based Text Editor

TO CONTINUE TYPE

Features

2500 line edit buffer
High speed CRT display
Command and input/change operating modes
Find/Replace command
UPPER/lower case alphabets
Block Move

This segment demonstrates the use of the Text Editor

to enter and change source files in

GenRad/futuredata

format

TO CONTINUE TYPE ^

JE

This command invokes the Editor.

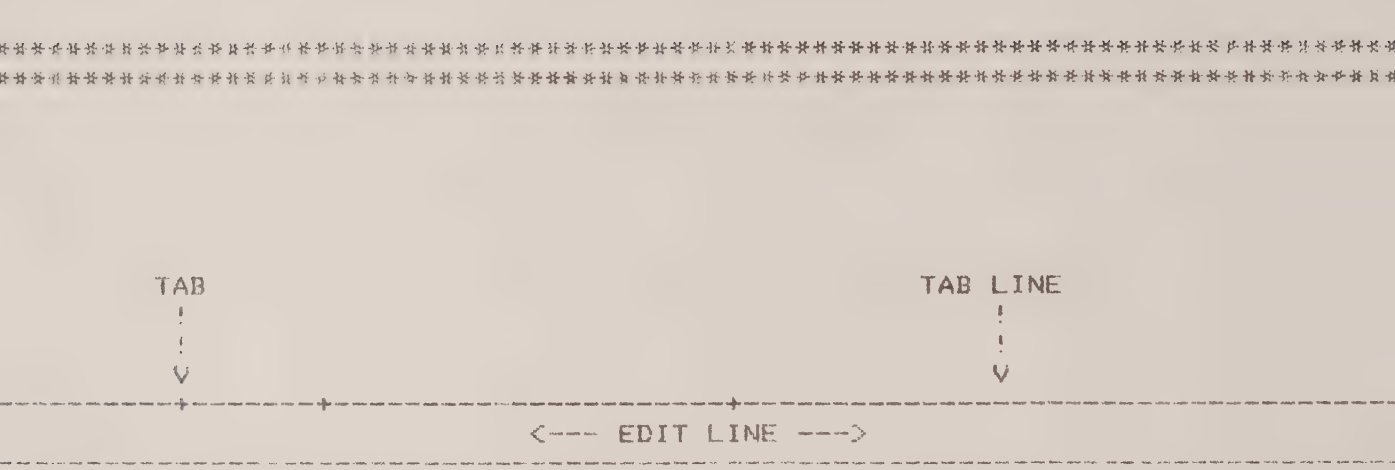
When ^ is entered, the module jumps to the Editor. Two lines are displayed in the middle of the screen and a blinking cursor is on the bottom line. The second line from the bottom of the screen displays the futuredata logo and the message "EDITOR V02" This line will disappear when the first command is entered in the Editor.

The Editor commands and format of the CRT are explained below

At this point the user is able to jump to any area in the operating system. Enter the command (JE<RETURN>) to jump to the Editor

To continue with this module, type ^. The format of the Editor display is explained in the next three pages.

EDITOR DISPLAY



Above is an Editor display with all of the areas labeled. The next four displays in this module define the terms used above. These terms are used throughout this module.

The next two pages have copies of the displays which explain the Editor format.

EDITOR DEFINITIONS

TAB LINE

The dashed line just above the center of the display is called the tab line. Vertical lines (+'s) on the tab line show tab stop positions.

TO CONTINUE TYPE ^

EDIT LINE

The center line of the display, between the two dashed lines, is the edit line. All text entry and editing is done on this line.

EDITOR DEFINITIONS

INSERT LINE

The insert line is the dashed line below the edit line. It is dashed in the Command and Line Edit modes and a double line (=) in the Line Insert mode.

TO CONTINUE TYPE ^

ENTRY CURSOR

The entry cursor is a blinking rectangle. This cursor is present at the command line during entry of commands and at the edit line during entry of text.

EDITOR DEFINITIONS

COMMAND LINE

The command line is the last line of the display. When the entry cursor (the blinking rectangle) is positioned on this line, commands to the Editor may be entered.

TO CONTINUE TYPE ^

MESSAGE LINE

This line is the second line from the bottom of the display. When an error is made in the entry of a command, an error message appears here. Some Editor commands request verification on this line.

EDITOR DEFINITIONS

WORKSPACE

The workspace is a part of computer memory which holds 2000 to 2500 lines of text (with 48K of memory). The text may have been entered from a diskette file or from the keyboard.

TO CONTINUE TYPE ^

DISPLAY

There is a special definition of the display during Editor operation. The display is that part of the workspace which is visible. The display is 24 lines high and 80 characters wide. 21 lines of text are visible, and 3 lines are used for the command line, the tab line, and the insert line.

****
****

LOAD EDITOR WORKSPACE FROM A FILE

L[filename]

This command loads text from the input file to the workspace If the edit buffer is not empty, the new information is appended to the end.

EXAMPLE NEXT, TO CONTINUE TYPE ^

If filename is omitted, text is loaded from the input file previously specified in an L command. If a filename is specified, that file then becomes the new input file. The first eleven lines of the loaded text are immediately visible on the display. Other areas can be displayed using commands described later in this module.

****
****

The first time an "L" command is executed, text is loaded from the begininning of the file and continues unitl the workspace is full or the end of the file is reached. Second and all subsequent loads load the workspace after the last line previously loaded in the edit buffer.

If an attempt is made to load into the workspace after the end of the file has been reached, the message "END FILE" is displayed

If the workspace is filled before the end of the file is reached, the message "FULL, MORE DATA IN FILE" is displayed.

When ^ is entered, the module enters the command "LINCR.N" on the Editor command line.

When <RETURN> is entered, the workspace is loaded with the short program INCR. This program will be used throughout the rest of the modules.

The user may now load a file into the workspace (for example LCOUNTER N) and this will be appended to the program INCR.

To continue, enter ^.

WRITE THE EDITOR WORKSPACE TO A FILE

W[$][*][file-name]

If filename is not specified, text will be written to the output file specified in the last W command. If the file has not yet been created, it must be prefixed by N:.

TO CONTINUE TYPE ^

[$] will write the workspace from the beginning up to but not including the text entry line to the output file.
[*] will cause the text written to the output file to remain in the workspace.
Both [$] and [*] may be used in the same command, but only in the order shown.

This command allows the user to save the information in the Editor workspace on disk. Whenever a write (W) command is executed, the Editor retains the name of the file specified and that name becomes the output file. This allows the user to append information to the ou ut file by entering a "W" command without a filename. If a write (W) command is entered without specifying a filename and without previously specifying an output file the message "FILE NOT SPECIFIED" will be displayed and the contents of memory will remain unchanged.

If an attempt is made to write to a new output file without ending the use of the old file, the message "END PREVIOUS FILE" is displayed. Typing "Y" closes the previous open file and typing "N" or <RETURN> cancels the Write command.

The $ parameter is used when the workspace is full and the user wants to save part of the workspace in memory. This command writes all text above edit line to disk.

The * parameter will allow the user to update the disk in the middle of a long edit session without doing a subsequent load. This provides the user with a non-volatile copy of the information in volatile memory. If there is a power failure or the power cord is accidently unplugged, memory data is not lost. This command will write all of the edit buffer out to disk, but also save it in memory.
To continue, enter ^.

WRITE EDITOR WORKSPACE AND THEN LOAD

N

TO CONTINUE TYPE ^

This command is equivalent to a W followed by an L. Both input and output filenames must have been previously specified.

This command is used when working with long text files. It is recommended that the user break assembly language programs into smaller modules to facilitate the debugging process. This will also eliminate the need for working with files requiring multiple loads and writes.

To continue, enter ^.

END THE OUTPUT FILE

E

TO CONTINUE TYPE ^

This command terminates the use of the previously specified output file.

The Editor allows the user to have one input and one output file open at a time. When the user wishes to write to a new file he must first close or END the exi tng output file

This command allows the user to END an output file. The Editor also allows the user to close an output file when writing to a new output file by typing Y in response to the prompt "END PREVIOUS FILE?". The Editor displays this prompt if the user tries to write to a file without closing the last output file

To continue, enter ^.

****

ADVANCE THE EDIT LINE

<DOWN ARROW>

This command advances the edit line by one position in the edit buffer. Holding this key down will scroll through the text in the edit buffer.

TO CONTINUE TYPE ^

A[n]

Advances edit line n lines in the edit buffer for 0<n<255.

A$

Advances edit line to the end of the edit buffer.

>

****

These commands allow the user to advance the edit line through the workspace.

The "n" in this command is the number of lines to be moved. It can be any number from 1 to 255.

To continue, enter ^.

Examples of the Advance command will follow the next display.

BACKUP THE EDIT LINE

<UP ARROW>

This command moves the edit line back one line in the edit buffer. Holding the key down will scroll the edit line through the text in the edit buffer.

EXAMPLE NEXT, TO CONTINUE TYPE ^

B[n]

Moves the edit line back n lines in the edit buffer for 0<n<255.

B$

Moves the edit line to the beginning of the edit buffer.

These commands allow the user to move the edit line toward the beginning of the workspace.

The "n" in this command is the number of lines to be moved It can be any number from 1 to 255.

After entering ^, the Editor loads the file named SCROLL. At this point the user may use the scroll keys (up and down arrows) to scroll through the text

After using the scroll keys (up and down arrows only), move the edit line so that line 0 is in the edit line.

After entering ^, the module displays the command A10.

After entering <RETURN>, the edit line advances to line 10 The module also displays the command B5.

After entering <RETURN>, the edit line scrolls back 5 lines to line 5. The module also displays the command A$.

After entering <RETURN>, the edit line is at the end of the workspace (one line after the last line of text) The module also displays the command B$.

After entering <RETURN>, the edit line moves to the beginning of the workspace (line 0) Command is now returned to the user

To continue, type ^.

DELETE LINES

DEL[n]

If n is not specified it defaults to 1.

EXAMPLE NEXT, TO CONTINUE TYPE ^

This command deletes the edit line and n-1 lines beneath the edit line toward the end of the workspace.

The "n" in this command is the number of lines to be deleted. It can be any number from 1 to 255.

This command allows the user to delete a specified number of lines from the workspace. Once the lines are deleted, however, they cannot be retrieved unless they are on the diskette.

When a ^ is entered, the module loads a file and the command DEL is entered.

When <RETURN> is entered, the line in the edit line (line 0) is deleted. The next command DEL5 is entered on the command line.

When <RETURN> is entered, the next 5 lines are deleted. The edit line now contains line 6. Control is now returned to the user.

To continue, enter ^.

CLEAR EDITOR WORKSPACE

CLR

This command clears the display and all memory assigned to editor workspace.

EXAMPLE NEXT, TO CONTINUE TYPE ^

Note that the CLR and the DEL commands that destroy the contents of the Editor workspace are three characters long. This is a protective measure so that text is not accidentally destroyed.

The CLR command is used to clear the Editor workspace. If the user tries to jump to another part of the operating system and the workspace is not empty, the Editor will prompt "SAVE DATA". If the user enters N, then the workspace is cleared and the jump is executed. Any other character aborts the jump command and the data in the workspace is not lost. This insures that the user will not accidentally lose the results of an edit session

To continue, enter ^.

RETURN will clear the Editor buffer.

THEN TYPE ^ TO CONTINUE

After typing the last "^" the module displays the above information and the CLR command. Typing <RETURN> will clear the workspace and leave the Editor display empty.

The user may try any of the commands previously explained before continuing the module.

To continue, enter ^.

SET AND CLEAR TABS

T[column-number][,column-number]...

Column-number is an integer between 2 and 80 inclusive. column-numbers must be in order (i.e. T10,5 is an error).

TO CONTINUE TYPE ^

The T command without parameters clears all tabs. Default Editor tabs are set in columns 11, 19 and 42 for entering assembly language programs. Each tab stop appears as a "+" symbol on the tab line.

Tabs can be set anywhere on the line. A tab is preset in the first column and cannot be removed

It is important to note that the tab locations must be specified from left to right. If the order is not correct, there will be a syntax error.

When a "T" command is entered, all existing tabs are cleared before the new tabs are set. Therefore a T command with no column numbers will clear all tabs.

To continue, enter ^.

SET AND CLEAR TABS

TT

This command sets the tabs in every eight column. This option is available to ease text entry.

EXAMPLE NEXT, TO CONTINUE TYPE ^

TA

This command sets tabs for assembly language programs. The tabs are set in their default locations in columns 11, 19 and 42.

>

The TT command sets tabs in every eighth column and is useful for writing text.

The TA command sets tabs at columns 11, 19, and 42. This is the initial setting when the Editor is invoked and is used for writing assembly language code.

At this point, the command line displays a T. This is the Tab command with no parameters. Upon entering <RETURN> the Editor clears all tabs except the one always set at column one.

Three examples will follow.

To continue, enter ^.

TYPE RETURN TO SET TABS IN COLUMNS 10, 20, 30, 40, 50, 60, 70.

FOR NEXT EXAMPLE TYPE ^

The command line now displays:

T10, 20, 30, 40, 50, 60, 70

Before typing <RETURN> there are no tabs on the tab line.

After typing <RETURN> tabs are set every 10 columns.

Any specified location can be set with a tab.

In this example, commas were used as delimiters between tab locations; a space is also an acceptable delimiter.

To continue, enter ^

```
TYPE RETURN TO SET TABS IN COLUMNS 11, 19 AND 42.

----------+---------+---------+---------+---------+---------+---------+---------
                        TO CONTINUE TYPE  ^
-------------------------------------------------------------------------------

THIS IS THE DEFAULT FOR TABS WHEN ENTERING THE EDITOR.
```

At this point the Editor displays TA on the command line.

Before typing <RETURN> there are tabs set every 10 spaces.

After typing <RETURN> tabs are set in columns 11, 19, and 42.
This is the default for tabs when the Editor is invoked.

To continue, enter ^.

TO CLEAR THE TABS TYPE <RETURN>

TO CONTINUE TYPE ^

At this point, the command line displays a "T". This is the tab command with no parameters. Upon entering <RETURN> the Editor will clear all tabs (except the one always set at column one).

Control now returns to the user. The user may now set tabs in any column

To continue, enter ^.

****
****

SET UPPER CASE MODE

SU

TO CONTINUE TYPE ^

SU causes all keyboard alphabetic entries to be entered as upper case only. This is useful for entering assembly language mnemonics. This is the default mode upon entering the Editor.

>
****
****
****

SET UPPER/LOWER CASE MODE

SL

TO CONTINUE TYPE ^

The SL command allows entry of both upper and lower case characters, as on a typewriter. This is useful for typing program comments or text.

>
****
****

****
****

UPPER/LOWER CASE TOGGLE

<CTRL> <C>

TO CONTINUE TYPE ^

When <CTRL> <C> is entered in any Editor mode, the keyboard is case toggled. This allows the user to switch from upper case to to upper/lower case and back in line edit mode without returning to the command line.

****
****

This feature allows the user to enter program comments in upper/lower case and yet be able to write program code in upper case without going back to the command line. When the <CTRL> <C> keys are entered simultaneously, the keyboard is toggled from upper case to upper/lower case or vice-versa, depending on the present state of the keyboard.

To continue, enter ^.

****
****

LINE INSERT MODE

I

This command allows the user to insert new lines of text between the present edit line and the line above the edit line.

TO CONTINUE TYPE ^

When this command is executed the present edit line and all lines below it are pushed down one line leaving the edit line empty except for the cursor. Characters may then be entered on the edit line from the keyboard. At the end of a line the <RETURN> key causes the entire process to be repeated.

>

****
****

After the next display, there will be an example of inserting lines into the Editor workspace.

To continue, enter ^.

LINE INSERT MODE

I

EXAMPLE NEXT, TO C'NTINUE TYPE ^

Two successive depressions of the <RETURN> key, with no intervening keystrokes exit the Line Insert mode and return the cursor to the command line for the next command.

****

When ^ is entered, the screen i cleared and then the workspace will be loaded with eight lines of text. The Editor also displays the I command.

When <RETURN> is entered, the Editor moves the present edit line down one line and enters a line of stars. Enter five successive returns. With each return, the module will enter a new line of text in the edit line. The next <RETURN> will result in a blank edit line. The next return (with no characters entered before the return) will cause the cursor to exit the Insert mode and return to the command line.

The user may now try the Line Insert mode and the upper/lower case toggle <CTRL> and <C> .

To continue, enter ^.

****
****

LINE EDITING MODE

X<RETURN> or <RETURN>

TO CONTINUE TYPE ^

A <RETURN> with no command on the command line moves the cursor to the edit line to enable editing. The effects of the special function keys are explained below.

>

****
****

The Line Editing mode and the special function keys allow the user to selectively change the program or text in the workspace.

To enter the Line Editing mode, the user types <RETURN>.
The cursor moves to the first column in the edit line.
The cursor control keys allow the user to non-destructively move through the workspace. All other keys will alter the workspace in some manner. The effects of the keys are expained below.

To continue, enter ^.

CHARACTER INSERT SUB-MODE

<INS>

Characters may be inserted or deleted using this submode without retyping the rest of the line.

TO CONTINUE TYPE ^

Depressing <INS> turns this mode on or off (a toggle switch). This submode is available in either LINE INSERT MODE (I) or LINE EDIT MODE (X). The Editor is in CHARACTER INSERT submode when there is an asterisk under the cursor in the edit line.

****

In Character Insert submode all characters on the line to the right of the cursor will move to the right the number of character spaces entered. The <BACK SPACE> key will remove only the character to the left of the cursor.

The Character Insert submode may be entered from the Line Insert mode or from the Line Edit mode.

The next seven displays are reproduced in the following four pages. These charts explain the effects of the special function keys in the Command Line mode, the Line Insert mode, the Line Editing mode and the Caharacter Insert submode.

To continue, enter ^.

****
****

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
<DEL>	No effect.	Removes character above cursor and moves entire line to right of cursor one space to left.	Same as LINE INSERT.	Same as LINE INSERT.
<CAN>	Remove entire line.	same	same	same

TO CONTINUE TYPE ^

>

****
****
****

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
<RETURN>	Enters command.	Depressing <RETURN> key twice causes Editor to leave Line Insert mode and returns to Command mode.	Editor leaves Line Editing mode, returns to Command mode.	Editor leaves Character Insert mode. Continues in Line Insert or leaves Line Edit mode.

TO CONTINUE TYPE ^

>

****
****

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
<UP ARROW>	Scrolls edit line up and blanks command line.	Same as <RETURN>.	Scrolls edit line up, cursor remains on edit line.	Leave Character Insert mode, scroll text up
<DOWN ARROW>	Scrolls edit line down and blanks command line	Same as <RETURN>.	Scrolls edit line down, cursor remains on edit line.	Leave Character Insert mode, scrolls edit line down.

TO CONTINUE TYPE ^

>

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
<RIGHT ARROW>	Spaces cursor to right, does not affect text.	same	same	same
<LEFT ARROW>	Spaces cursor to left, does not affect text.	same	same	same

TO CONTINUE TYPE ^

>

****

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
<SPACE>	Enters a blank under cursor, moves cursor one space to right.	same	same	Moves cursor, character under cursor, and enti re line to right of cursor one space to right. Inserts blank at former cursor position.

TO CONTINUE TYPE ^

>

****

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
<BACK SPACE>	Moves cursor left one space, re moves character under cursor and one character to the left.	same	same	Removes characte acter to left of cursor. Cursor and entire line to right of cur- sor move one space left. Adds blank on right end of line.

TO CONTINUE TYPE ^

>

****

SPECIAL KEYBOARD EDITING OPERATIONS

	COMMAND MODE	LINE INSERT MODE	LINE EDIT MODE	CHARACTER INSERT
INS	Enters character corresponding to key.	same	same	Moves cursor, character under cursor, and all of line to the right. Inserts character corresponding to key at former cursor position

TO CONTINUE TYPE ^

>

This is the end of the tables explaining the effects of the special function keys in the Editor.

To continue, enter ^.

****
****

FIND (AND REPLACE) A STRING IN WORKSPACE

F[{delim}{string}{delim}[{rstring}{delim}[/[A][V]]]]

This command searches for a match between string and a character or series of characters in the workspace. The command may display the first line containing a match. Also, it may replace string with rstring in the first line or all lines containing a match.

TO CONTINUE TYPE ^

---

delim is any character occurring in neither string or rstring.
string is any series of display characters, including blanks.
If string is not specified, the previously specified string is used.
A (all) specifies that all matches of string from the current edit line to the end of the workspace be replaced.
V (verify) specifies that the system ask for verification before replacing a string. Type Y to replace, <RETURN> to continue without replacing and N to terminate F command.

>

****
****

There are two more displays on the FIND AND REPLACE command and several examples.

To continue, enter ^.

****

****

FIND (AND REPLACE) A STRING IN WORKSPACE

FINDING A STRING

When this command is used to find a string of characters in the workspace, searching begins on the first line following the current edit line. The first line in which a match occurs is displayed.

TO CONTINUE TYPE ^

Since the parameters of the previous F command are saved, subsequent searches may be initiated by executing the command F<RETURN>. After a line is displayed, it is possible to enter the line edit mode and change the text on the line or any of the surrounding lines.

>

****

****

To continue, enter ^.

****
****

FIND (AND REPLACE) A STRING IN WORKSPACE

FINDING AND REPLACING A STRING

When this command is used to replace a string of characters, searching begins on the current edit line. If A is specified, searching and replacing continues to the end of the workspace.

EXAMPLE NEXT, TO CONTINUE TYPE ^

It is possible to delete characters from lines in the workspace by specifying rstring as a null string by typing the second and third delimiters with no intervening characters or spaces.

****
****

When ^ is entered, the Editor loads INCR.N into the workspace and the command F/START/ is displayed. When <RETURN> is entered, the Editor finds the first occurrence of START in the file. This line is moved to the edit line. The module then displays F on the command line. When <RETURN> is entered, the Editor finds the next occurrence of START in the file.

Next, enter ^. This moves the edit line to the beginning of the program and displays the command F/START/BEGIN/A. When <RETURN> is entered, the Editor replaces all occurences of START with BEGIN and moves the edit line to the end of the workspace.

Next, enter ^. This again moves the edit line to the beginning of the program and enters the command F/BEGIN/START/AV. When <RETURN> is entered, the Editor finds the first occurrence of BEGIN in the file and asks the user if the line is to be replaced. The user may then enter Y if he wants to replace the string or <RETURN> if this occurrence is not to be replaced but the search is to continue. N will not replace BEGIN and will abort the Find Command.

After the cursor returns to the command line enter ^ to continue.

****

GET (AND REPLACE) A STRING IN FILE

G[{delim}{string}{delim}[{rstring}{delim}[/[A][V]]]]

This command searches for a match between string and a character or series of characters in the workspace and the input file. The command displays the first line containing a match. Also, it may replace the character string with rstring in the first line or in all lines containing a match.

TO CONTINUE TYPE ^

The G command is identical to the F command except that it searches both the workspace and the input file. Searching begins as in the F command but if the end of the workspace is reached during the search, all text in the workspace is written to the output file. The workspace is again filled with text from the input file and the search continues until a match between the text and the string is found or the end of the input file is reached.

>

****

This command is used when searching a file which is larger than the workspace.

To continue, enter ^.

****
****

BLOCK MOVE COMMANDS

The Block Move commands allow the user to copy a portion of the Editor workspace to the disk, copy a file from the disk to any location in the workspace, copy a block of text to another location in the workspace and delete a portion of the workspace.

TO CONTINUE TYPE ^

>

****
****

PLACE MARKER COMMAND

P

This command places a temporary marker in the Editor workspace just before the current edit line. When this command is entered, a solid line appears above the current edit line. Two markers delimit a block of text for subsequent copying, moving or deletion.

EXAMPLE NEXT, TO CONTINUE TYPE ^

When a second pointer is set, the workspace between the two pointers is then displayed in reverse video. Only two pointers may be present at one time If a third marker is entered, the first marker is automatically cleared.

These markers do not affect the workspace. They are cleared whenever the workspace is altered.

>

When ^ is entered, the Editor module loads a file called TREE. The command P is displayed on the command line to set the first marker.

When <RETURN> is entered, a marker (solid line) is placed just above the edit line. The comma . A3 is displayed on the command line.

When <RETURN> is entered, the edit line advances 3 lines. The command "P" is displayed on the command line.

When <RETURN> is entered, the second marker is placed just above the edit line (the fourth line in the file) and the first three lines are in reverse video. The command A3 is displayed on the command line.

When <RETURN> is entered, the edit line advances 3 lines. The command P is displayed on the command line.

When <RETURN> is entered, lines four to six in the file are in reverse video. The command B5 is displayed on the command line

When <RETURN> is entered, the edit line backs up 5 lines. The command P is displayed on the command line.

When <RETURN> is entered, five lines are now in reverse video. The user may now set pointers in other areas.

To continue, enter ^.

****
****

COPY COMMAND

C

The Copy command inserts the block of text defined by two markers just above the current edit line. The original block of text is not modified or deleted and the markers are not cleared.

EXAMPLE NEXT, TO CONTINUE TYPE ^

A block of text may be copied repeatedly to any location in the workspace except within the reverse video block of text by entering a series of C commands.

If the workspace is filled, the message OVERFLOW is displayed.

The C command does not clear the markers so they can be used repeatedly.

>

****
****

When ^ is entered, the file TREE is loaded, pointers are placed around the first three lines and the edit line is advanced to the seventh line in the workspace. The C command is displayed on the command line.

When <RETURN> is entered, the first three lines are copied just above the edit line and the C command is again displayed.

When <RETURN> is entered, the three lines are copied again. The command B1 is displayed on the command line.

When <RETURN> is entered, the edit line backs up 1 line. The command C is displayed on the command line.

When <RETURN> is entered, the three lines between the markers are again copied. The command B5 is displayed on the command line.

When <RETURN> is entered, the edit line backs up 5 lines. The user is now free to set pointers and copy blocks of text.

To continue, enter ^.

COPY FROM COMMAND

CF[filename]

The Copy From command allows the user to insert a file from the disk to any location in the workspace. The file is inserted just prior to the current edit line analogous to the Insert and Copy commands.

EXAMPLE NEXT, TO CONTINUE TYPE ^

The first time a CF command is entered, a filename must be specified. If the workspace is filled during the copy, the message OVERFLOW is displayed. the user may then save part of the workspace using the W$ command and may then complete the transfer by entering a CF command without a filename.

>

When ^ is entered, the module is loads the file TREE and advances the edit line to the fifth line in the file. The command CFINSERT is displayed on the command line.

When <RETURN> is entered, the file INSERT will be copied just above the edit line. This file is five lines and has the message INSERTING DATA FROM ANOTHER FILE.

The user is now free to try the CF command.

To continue, enter ^.

****

COPY TO COMMAND

CT[filename]

The Copy To command copies the block of text defined by two markers to the specified disk file. When a filename is specified, the file is overwritten. The CT command does not alter the workspace nor does it clear the markers.

EXAMPLE NEXT, TO CONTINUE TYPE ^

The first time a CT command is entered, a filename must be specified. Subsequent CT commands without a filename will append the blocks of text to the file.

The CT command does not affect the W, L, or N commands.

A CT command may not be entered without a filename after a CF command has been entered.

>

****

When ^ is entered, the file TREE is loaded and markers are placed after the first and fourth lines in the file. The command CTSAVE is displayed on the command line.

When <RETURN> is entered, the three lines between the markers are saved on the disk but the workspace is not altered. The command CLR is displayed.

When <RETURN> is entered, the workspace is cleared and the command LINSERT is displayed.

When <RETURN> is entered, the file insert is loaded and the command A2 is displayed.

When <RETURN> is entered, the edit line advances 2 lines and the command CFSAVE is displayed.

When <RETURN> is entered, the file SAVE with the three lines of asterisks is copied just above the edit line.

The user is now free to use the CT and CF commands.

To continue, enter ^.

****

KILL COMMAND

K

The Kill command deletes the block of text defined by two markers and clears the markers.

EXAMPLE NEXT, TO CONTINUE TYPE ^

If the current edit line lies within the deleted block of text, the first line after the reverse video block of text becomes the new edit line.

>

****

When ^ is entered, the file TREE is loaded and markers are placed below lines one and five The K command is displayed on the command line.

When <RETURN> is entered, the four lines in reverse video are deleted from the workspace.

The user is now free to place markers and use the K command

To continue, enter ^.

****
****

MOVE COMMAND

M

The Move command inserts the block of text defined by two markers just above the current edit line, deletes the original block of text and clears the markers. The M command is equivalent to a C command followed by a K command.

EXAMPLE NEXT, TO CONTINUE TYPE ^

The M command may be executed when the workspace is full because a text rotation algorithm is used. This command allows the user to rearrange the contents of the workspace.

>

****
****

When ^ is entered, the file TREE is loaded and markers are placed below the second and fifth lines. The edit line is advanced to the ninth line in the file. The M command is displayed on the command line.

When <RETURN> is entered, the lines in reverse video are inserted above the edit line and are deleted in their original position.

The user is now free to place markers and use the "M" command or any of the other Block Move command.

To continue, enter ^.

****

Command is now returned to the user to try features of the GENRAD/DSD Editor. Refer to the reference card for a summary of commands.

TO CONTINUE WITH SELF-INSTRUCTION PACKAGE TYPE ^

>

****

To try some of the commands of the Editor, first clear the screen. This is done by entering CLR<RETURN>. Input of text in the Editor can be done by the Load or the Insert command Do not enter ^ unless you are ready to go to the next module.

To start the next module on the Assembler type ^ to call up the menu. Then enter S3<RETURN> to invoke the Assembler module.

****
****

GenRad/DSD

Advanced Development System

ASSEMBLER

TO CONTINUE TYPE ^

Features:

Output is relocatable file
Menu driven
Self-prompting
Allows conditional assembly
Allows macro libraries

>

****
****

```
JA

This command invokes the Assembler.

-------------------------------------------------------------------------------
                        TO CONTINUE TYPE ^
-------------------------------------------------------------------------------

A list of Assembler options will then
appear on the screen.

>
```

After typing ^, the module displays JA on the command line.
This command invokes the Assembler

To continue with the module and an explanation of the
Assembler options, type a ^ from the assembler display.

The following page and two displays explain the Assembler
options. Enter successive ^'s for the explanations.

ASSEMBLER OPTIONS

L — Display a program listing on the CRT. This option is affected by the T and E options.

T — Truncates lines of the display to 80 characters. Printer lines are never truncated. The T option also displays only one line for each DC statement instead of one line per byte.

E — Only lines containing errors flagged by the Assembler are displayed if L or T is specified or will be sent to the listing file.

TO CONTINUE TYPE ^

>

ASSEMBLER OPTIONS

S — This option includes a table of symbolic labels at the end of the binary object file. During debugging memory locations can be referenced by their label names if a symbol table is written by the Assembler, selected as a Linker option and loaded by the Debugger.

Z — Flag all lines containing Z80 instructions which have no 8080 equivalent with the message: INVALID OPCODE. This option is useful for verifying Z80 and 8080 program compatibility.

TO CONTINUE TYPE ^

>

****

ENTRY OF ASSEMBLER OPTIONS

The Assembler options are entered immediately after the list of options.

TO CONTINUE TYPE ^

There should be no intervening blanks between Assembler options. <RETURN> causes the Assembler to accept the options selected.

>

****

The options may be specified in any order. The user may jump to another part of the operating system only from this line.

Some common selections of Assembler options are below:

IS	This is the usual selection if there is no printer. It sends a truncated listing to the CRT and includes the symbol table for symbolic debugging.
ES	This set of options prints only the lines containing errors and the error messages to the CRT and/or listing file. The symbol table is included for symbolic debugging.

To continue, enter ^.

****
****

SOURCE FILE:

After entering Assembler options the Assembler asks for a source file as the input file to the Assembler.

TO CONTINUE TYPE ^

This file must have an S attribute and may be on either Drive 0 or Drive 1. Filename should be terminated by <RETURN>.

>

****
****

The Assembler always prompts the user for a source file after the options have been specified. This file must have an "S" attribute. It should have no more than eight relocatable segments of code. It may also have macro definitions at the beginning of the file before any other code.

If N: precedes the filename, it will be ignored. The other filename conventions specifying the filename and the drive number are applicable.

To continue, enter ^.

****

MACRO FILE:

After entering Assembler options and the name of the source file, the Assembler will prompt for the name of a macro file. This input is optional. Enter <RETURN> if no macro file is used.

TO CONTINUE TYPE ^

This file must have an S attribute and may be on either Drive 0 or Drive 1. The filename should be terminated by <RETURN>. This file is searched for a macro definition if the macro is not defined in the source file. The end of the macro file is the first occurrence of a statement other than a printer control directive after a macro definition.

>

****

The macro file must have an "S" attribute. The macros must start at the beginning of the file and there must be no code or comments between macro definitions. The end of the macro file is defined as the first occurence of a statement other than a macro definition. Comments between macro definitions are illegal, but comments within macros are legal. This definition also allows the user to define another program with macros at the beginning of the file to be the macro file.

The macro file allows a group of users to use a common macro library. If the user does not want to use a macro library but still wants to use macros, they must be defined at the start of the source file. Macros defined at the beginning of the source file take precedence over macros defined in a macro file.

To continue, enter ^.

****
****

OUTPUT FILE:

After entering the Assembler options, the source file and the macro file, the Assembler will prompt for the output file. This input is optional. Enter <RETURN> if no output code file is desired.

TO CONTINUE TYPE ^

This file must have an R attribute. If a new file is created (N:), it will be given an R attribute. The output file may be on either drive, but the Assembler is much faster if the output file for the relocatable code is on a different drive from the source input file. The filename should be terminated by <RETURN>.

>

****
****

When an output file is specified, the code file will always be created, even if there are errors in the assembly.

The output file must not have a W attribute or the message WRITE PROTECTED FILE will be displayed.

The Assembler is substantially faster if the output file is on a different drive from the source file. Using two disk drive heads decreases the time necessary to position the heads for a read or a write.

To continue, enter a ^.

****

LISTING FILE:

After entering the Assembler options, the source file, the macro file and the output file, the Assembler will prompt for the listing file. This input is optional. Enter <RETURN> if no listing file is desired.

TO CONTINUE TYPE ^

Enter P: to have the assembler listing sent to the Centronics compatible printer port. Enter a filename and the listing will be spooled on the disk. It may have no attribute or an S attribute. This file may be loaded into the Editor for viewing.

>

****

When a listing file is specified, any information sent to the CRT is sent to the listing file. Lines sent to the listing file are not truncated to 80 characters if the T option was specified.

If no printer is attached to Printer port, the message PRINTER NOT READY is displayed and the Assembler again prompts for a listing file.

When a filename is specified, the listing is saved on the disk. This file may then be sent to the printer at a later time. If the file is given an S attribute, it may be loaded into the Editor. There are be several printer control characters in the file which make the file not completely Editor compatible, but the file may be viewed in the Editor and errors may be easily located by finding a string of two asterisks (**).

To continue, enter ^.

****
****

LISTING FILE:

S{n}:[/BAUD=300][/PARITY=NONE][/LENGTH=7][/STOP=1]

This response sends the assembler listing out the specified serial port and allow the alteration of default values for the serial port parameters.

TO CONTINUE TYPE ^

****
****

The listing may be sent out a serial port not only for listing on a serial printer, but it may also be sent out a port so that the listing is slowed down on the CRT for easier viewing. For instance, entering S2:/BAUD=2400 will display the listing on the CRT only as fast as the listing is sent out the serial port.

The standard serial port routines do not have any hand-shaking. The source for the serial port routines is located on the Utility disk. The serial drivers may be modified and then linked into the Assembler by using the Assembler configuation disk.

To continue, enter ^.

****

The GenRad/DSD
is a two pass Assembler.

The message PASS1 is displayed when the Assembler starts the first pass.

TO CONTINUE TYPE ^

The message PASS2 is displayed when the Assembler starts the second pass.

When the Assembler is finished, it waits for <RETURN> before it clears the screen and lists the Assembler options again. A jump command or Assembler options for another assembly may then be entered.

>

****

During the first pass, the Assembler is setting up a symbol table for the second pass when the code is actually generated.

Symbols are either local, global or external.

A local symbol is one that is defined and referenced in only one RSEG (relocatable segment of code). A global symbol is either an entry point for the RSEG or a label defined in another RSEG. An external symbol is a symbol defined in another file but is used in the file being assembled.

In the symbol table G means the symbol is a global symbol and X means it is an external symbol, Local symbols have no special designation.

To continue, enter ^.

****
****

RESTARTING THE ASSEMBLER

During the second pass typing any key halts the Assembler.
The message CONTINUE? is displayed.

TO CONTINUE TYPE ^

Y, <RETURN> or <SPACE> restart the Assembler.
N, <BREAK> or <RESET> aborts the assembly and any other character causes the message CONTINUE? to be displayed again.

>
****
****

Since the Assembler displays the listing to the CRT very quickly, the user may temporarily halt the Assembler to examine an error which has been flagged. This option is exercised when the user is not sending an assembler listing to a printer.

To continue, enter ^.

****
****

Following are two examples using the Assembler.

TYPE <RETURN> TO STEP THROUGH THE EXAMPLES

TO CONTINUE TYPE ^

The Assembler is slowed down for the first example for ease of viewing the output on the screen. The Assembler is running at full speed for the second example.

>

****
****

After entering ^, the module jumps to the Assembler and enter "TS" as the assembler options. This generates a truncated listing to the CRT and includes the symbol table in the object file.

Enter <RETURN>. The module displays INCR.N as the source file.

Enter <RETURN>. The module does not display a filename for the macro file.

Enter <RETURN>. The module displays N:1:INCR.R as the output file where the assembler will store the relocatable object code.

Enter <RETURN>. The module displays S2:/BAUD=1200 as the listing file. This sends the listing out serial port 2 at 300 baud.

Enter <RETURN>. The module assembles the program at this time. The next page shows the display after the above actions and also shows the assembler listing.

```
SPECIFY ASSEMBLER OPTIONS:
 (L) - LIST ON SCREEN
 (T) - TRUNCATED LIST
 (S) - INCLUDE SYMBOL TABLE
 (E) - LIST ERRORS ONLY
 (Z) - FLAG NON-8080 OPCODES
>TS
SOURCE FILE:
>INCR.N
MACRO FILE:
>
OUTPUT FILE:
>N:1:INCR.R
LISTING FILE:
>S2:/BAUD=300
PASS 1
PASS 2
                *                                THIS PROGRAM IS INTENDED TO BE USED
                *                                WITH THE Z80 SELF-INSTRUCTION
                *                                PACKAGE FOR THE GENRAD 2300 ADS.
 0000                       RSEG    INCR         RELOCATABLE PROGRAM SEGMENT
 0000 3C         START      INC     A
 0001 D300                  OUT     (X'0'),A     OUTPUT TO PORT 0
 0003 C30000                JP      START
 0006                       END     START
 INCR      G 0000      START       0000
NO ERRORS
```

For the next example enter <RETURN>. The Assembler will returns to the original display and he module enters TS for the assembler options.

Enter <RETURN>. The module then displays INCR.N as the source file.

Enter <RETURN>. The module does not display a macro filename.

Enter <RETURN>. The module then displays 1:INCR.R as the relocatable code file.

Enter <RETURN>. The module does not enter a listing file.

Enter <RETURN> and the Assembler reassembles the same segment of code. This time however, the Assembler operates at full speed. Notice how much faster the Assembler works when it is not slowed down to send the listing out the serial port at 300 baud.

Enter <RETURN> and the module scratches the file INCR.R on Drive 1 before returning to the menu. This must be done or the module will not execute the next time it is called.

The module then displays the menu to allow selection of the next feature.

****
****

GenRad/DSD

Advanced Development System

LINKER

TO CONTINUE TYPE ^

>

****
****

****
****

JL

This command invokes the Linker.

TO CONT NUE TYPE ^

A list of Linker options then appears on the screen.

>
****
****

After entering ^, the modul displays JL on the command line.
This invokes the Linker.

To continue with the module enter ^ from the Linker display.

The next page and display explain the Linker options.

```
**********************************************************************************
**********************************************************************************

                                LINKER OPTIONS
==================================================================================

        D       Only RSEG's specifically named in the command file or
                entered via keyboard are input to the Linker; others
                are deleted.  If D is selected, unspecified RSEG's are
                deleted from both the binary and display output.

        S       The symbol tables (which may be placed at the end of
----------------------------------------------------------------------------------
                each RSEG by the Assembler) are written at the end of
----------------------------------------------------------------------------------
                the absolute binary object file. These tables are required
                for symbolic debugging.

        L       Display the memory map and reference list.

        Z       Flag all Z80 instructions not translatable to 8080 code.

                            TO CONTINUE TYPE ^
>
**********************************************************************************
**********************************************************************************


        If the D option is not specified, all RSEG's not explicitly
        mentioned, will be linked into memory after the last RSEG named.


        To continue, enter ^.
```

****

ENTRY OF LINKER OPTIONS

The linker options are entered immediately after the menu.

TO CONTINUE TYPE ^

There should be no intervening blanks between Linker options.
<RETURN> causes the Linker to accept the options selected.

>

****

The options may be specified in any order. This is also the only line from which the user may jump to another part of the operating system.

Some common selections of Linker options are listed below.

LS — This selection will is the most common sends a Linker memory map to the CRT and includes the symbol table for symbolic debugging.

DLS — This selection not only sends the memory map to the CRT and includes the symbol table, but also discards any RSEG's not explicity mentioned. This could reduce the size of the file.

To continue, enter ^.

****
****

SPECIFY INPUT FILE

After entering Linker options the assembler asks for an input file. Input files should have an R attribute.

TO CONTINUE TYPE ^

File name should be terminated by <RETURN>.

The Linker then prompts for another input file. <RETURN> in response to SPECIFY INPUT FILE signifies that there are no more input files.

>

****
****

After specifying the Linker options, the Linker prompts for an input file. The user should then enter the name of one of the files to be linked to form an absolute module. The file must have an R attribute. After specifying an input file, the Linker opens that file and reads the name and length of each segment of code in the file. If the L option was selected, this information is then displayed on the CRT. The Linker then prompts for another input file.

This sequence is repeated until <RETURN> is entered in response to the prompt. This informs the Linker that there are no more input files.

If N: precedes the filename, it is ignored. The other filename conventions specifying the filename and the drive number are applicable.

To continue, enter ^.

****

SPECIFY OUTPUT FILE

After entering the Linker options and the input files, the Linker prompts for the output file. This input is optional. Enter <RETURN> if no output file is desired.

TO CONTINUE TYPE ^

This file must have an O attribute. If a new file is created (N:), it will be given an O attribute. The output file may be on either drive, but the Linker is much faster if the output file for the absolute object code is on a different drive from the relocatable input files. The filename should be terminated by a <RETURN>.

>

****

The output file must not have a W attribute or the Linker will return the message WRITE PROTECTED. When an output file is specified, a file will always be created, even if there are errors in the link process.

The Linker is substantially faster if the input files are all on the same drive and the output file is on the other drive. Using two disk drive heads decreases the time necessary to position the heads for a read or a write.

To continue, enter ^.

****
****

LISTING FILE:

After entering the Linker options, the input files and the output file, the Linker prompts for the listing file. This input is optional. Enter <RETURN> if no listing file is desired.

TO CONTINUE TYPE ^

Enter P: to send the Linker memory map to the Centronics compatible printer port. Enter a filename and the map will be spooled on the disk. This file may have no attribute or an S attribute. It may be loaded into the Editor for viewing.

>

****
****

When a listing file is specified, any information sent to the CRT is sent to the listing file.

If no printer is attached to the Printer port, the message PRINTER NOT READY is displayed and the Linker again prompts for a listing file.

When a filename is specified, the listing is saved on the disk. This file may be sent to the printer at a later time. If the file is given an S attribute, it may be loaded into the Editor. However, there will be several printer control characters in the file which make the file not completely Editor compatible. The file may be located by finding a string of two asterisks (**).

To continue, enter ^.

****
****

LISTING FILE:

S{n}:[/BAUD=300][/PARITY=NONE][/LENGTH=7][/STOP=1]

This response sends the Linker memory map out the specified serial port and allows the alteration of the default values for the serial port parameters.

TO CONTINUE TYPE ^

>

****
****

The listing may be sent out a serial port not only for listing on a serial printer, but also to slow the listing file down on the CRT for easier viewing. For instance, entering S2:/BAUD=2400 displays the listing on the CRT only as fast as the listing is sent out the serial port.

The standard serial port routines do not have any hand-shaking. The source for the serial port routines is located on the Utility disk. The serial drivers may be modified and then linked into the Assembler by using the Linker configuation disk.

To continue, enter ^.

****
****

LINKER INPUT

Linker input commands specify names of RSEG's to be linked, the address at which the RSEG's are to be loaded into memory and the entry point for execution.

TO CONTINUE TYPE ^

>
****
****

After the user enters the name of the listing file, the Linker prompts for Linker input. Here the user specifies the location in memory for the RSEG's in the input files.

The next four displays and the next two pages explain the possible responses to the prompt for Linker input.

Note: Before any RSEG's are linked, the Linker will put all ASEG's into the proper memory locations.

To continue, enter ^.

****

#ORG {absolute-address}

This statement specifies the absolute address of the start of the first RSEG.

TO CONTINUE TYPE ^

This command must be followed by <RETURN>. Absolute address can be in two forms:

```
xxxxH      (Intel convention)
X'xxxx'    (IBM convention)
```

"X" is a hex digit. Leading zeros are automatically prefixed if necessary.

>

****

```
{rseg-name-a}[,rseg-name-b]...
[rseg-name]...
        .
        .
        .
```

TO CONTINUE TYPE ^

After specifying an address, specify a list of as many RSEG's as desired to be loaded in the given order starting at the address specified in the #ORG statement.

>

****

More starting addresses, each with a list of RSEG's, may then be specified.

TO CONTINUE TYPE ^

>

#END [global-name]

This statement ends the linker input. One and only one #END command may be used.

TO CONTINUE TYPE ^

---

global-name is used as the entry point of the program. If none is specified, the first byte of the first RSEG is the entry point.

>

****
****

Following are two examples using the Linker.

TO CONTINUE TYPE ^

TYPE <RETURN> TO STEP THROUGH EXAMPLES.

>
****
****

The first example links the small program in the source file INCR.N to run starting in location 300H in memory.

After entering ^, the module jumps to the Linker and displays LS as the Linker options. This generates a listing of the link map to the CRT and includes the symbol table for symbolic debugging.

Enter <RETURN>. The module then displays INCR.R as the input file.

Enter <RETURN>. The Linker shows that INCR.R has one RSEG called INCR which is 6 bytes long and prompts for the next input file.

Enter <RETURN>. The Linker prompts for an output file and the module responds with N:1:INCR.

Enter <RETURN>. The Linker does not display a listing file.

Enter <RETURN>. The module then displays #ORG 300H in response to LINKER INPUT.

Enter <RETURN>. The module displays INCR for the RSEG to be linked starting at location 300H.

Enter <RETURN>. The Linker shows that at location 300 is the RSEG INCR which comes from the file INCR.R and is 6 bytes long. The module enters #END.

Enter <RETURN>. The Linker then performs the linking function.

Shown below is the information displayed on the CRT.

```
SPECIFY LINKER OPTIONS:
 (D) - DELETE UNSPECIFIED RSEGS
 (S) - INCLUDE SYMBOL TABLE
 (L) - LIST TABLES ON CRT
>LS
SPECIFY INPUT FILE
>INCR.R
RSEG      LENGTH
INCR       0006
SPECIFY INPUT FILE
>
SPECIFY OUTPUT FILE
>N:1:INCR
SPECIFY LISTING FILE
>
LINKER INPUT
>#ORG 300H
>INCR
0300  INCR      INCR.R      0006
#END

FILE        RSEG       ADDR LENGTH
INCR.R      INCR       0300 0006

            GLOBALS
            INCR      0300


ENTRY POINT 0300
FUNCTION COMPLETED
```

To continue, enter <RETURN>.

Enter successive <RETURN>'s to step through the second example.

This example uses the same linker options LS but has two input files COUNTER.R and TIMER.R. The output file is 1:COUNTER. The three RSEG's are linked at locations 100H, 200H and 300H.

Shown below is the information displayed on the CRT.

```
LINKER DISPLAY
      >LS
      SPECIFY INPUT FILE
      >COUNTER.R
      RSEG      LENGTH
      COUNT      0012
      DATA       0001
      SPECIFY INPUT FILE
      >TIMER.R
      RSEG      LENGTH
      TIME       000E
      SPECIFY INPUT FILE
      >
      SPECIFY OUTPUT FILE
      >N:1:COUNTER
      SPECIFY LISTING FILE
      >
      LINKER INPUT
      >#ORG X'100'
      >COUNT
      0100  COUNT     COUNTER.R  0012
      >#ORG 200H
      DATA
      0200  DATA      COUNTER.R  0001
      >#ORG 300H
      >TIME
      0300  TIME      TIMER.R    000E
      #END

      FILE         RSEG       ADDR LENGTH
      COUNTER.R    COUNT      0100 0012
                   DATA       0200 0001

                   GLOBALS
                   COUNT    0100      DATA      0200

      FILE         RSEG       ADDR LENGTH
      TIMER.R      TIME       0300 000E

                   GLOBALS
                   TIME     0300      TIMEO     0300

      ENTRY POINT 0100
      FUNCTION COMPLETED
```

To continue with the module enter <RETURN>.

****
****

NOTES

If D Linker option is not selected all RSEG's not specified by name in response to the Linker input prompt are included after the last RSEG in both the binary and display output.

EXAMPLE NEXT, TO CONTINUE TYPE ^

If there is more than one RSEG of the same name in the input files, the Linker uses the first one encountered.

>

****
****

After entering ^, the Linker option menu is displayed with the LS options next to the cursor. The D option is not selected.

Enter successive <RETURN>'s to step through this example.

The two files, COUNTER.R and TIMER.R are the input files. The RSEG COUNT is linked at location 100, and since the other RSEG's are unspecified, they are linked in the following COUNT.

The display for this example is on the next page.

```
SPECIFY LINKER OPTIONS:
 (D) - DELETE UNSPECIFIED RSEGS
 (S) - INCLUDE SYMBOL TABLE
 (L) - LIST TABLES ON CRT
>LS
SPECIFY INPUT FILE
>COUNTER.R
RSEG      LENGTH
COUNT      0012
DATA       0001
SPECIFY INPUT FILE
>TIMER.R
RSEG      LENGTH
TIME       000E
SPECIFY INPUT FILE
>
SPECIFY OUTPUT FILE
>1:COUNTER
SPECIFY LISTING FILE
>
LINKER INPUT
>#ORG X'100'
>COUNT
0100  COUNT     COUNTER.R  0012
>#END
0112  DATA      COUNTER.R  0001
0113  TIME      TIMER.R    000E


FILE         RSEG       ADDR LENGTH
COUNTER.R    COUNT      0100 0012
             DATA       0112 0001

             GLOBALS
             COUNT    0100      DATA       0112


FILE         RSEG       ADDR LENGTH
TIMER.R      TIME       0113 000E

             GLOBALS
             TIME     0113      TIME0      0113


ENTRY POINT 0100
FUNCTION COMPLETED
```

To continue with the module enter <RETURN>.

****
****

NOTES

It is possible to define the orgin of an RSEG at the location of the previous RSEG using #ORG commands. If this is done, the overlaid RSEG is flagged by an O in the length column of the memory map.

EXAMPLE NEXT, TO CONTINUE TYPE ^

>

****
****

After entering ^, an example of an RSEG overlaying a previous RSEG is shown.

Enter successive <RETURN>'s to step through the example.

The display of the link map for this example is shown on the next page.

```
SPECIFY LINKER OPTIONS:
 (D) - DELETE UNSPECIFIED RSEGS
 (S) - INCLUDE SYMBOL TABLE
 (L) - LIST TABLES ON CRT
>LS
SPECIFY INPUT FILE
>COUNTER.R
RSEG      LENGTH
COUNT      0012
DATA       0001
SPECIFY INPUT FILE
>TIMER.R
RSEG      LENGTH
TIME       000E
SPECIFY INPUT FILE
>
SPECIFY OUTPUT FILE
>1:COUNTER
SPECIFY LISTING FILE
>
LINKER INPUT
>#ORG X'100'
>COUNT
0100  COUNT     COUNTER.R  0012
>#ORG 100H
>DATA
0100  DATA      COUNTER.R  00001
>#ORG 100H
>TIME
0100  TIME      TIMER.R    0000E
>#END


FILE         RSEG       ADDR LENGTH
COUNTER.R    COUNT      0100 0012
             DATA       0100 0001

             GLOBALS
             COUNT    0100      DATA       0100


FILE         RSEG       ADDR LENGTH
TIMER.R      TIME       0100 000E

             GLOBALS
             TIME     0100      TIMEO      0100


ENTRY POINT 0100
FUNCTION COMPLETED
```

Enter <RETURN> to continue the module.

****
****

NOTES

If no #ORG statement is specified before the first RSEG is specified, that RESG is loaded starting at location 0.

EXAMPLE NEXT, TO CONTINUE TYPE ^

>

****
****

After entering ^, the module displays an example of an RSEG being linked starting at location 0 because the initial ORG statement was not included as Linker input.

The example uses the file INCR.R.

Enter successive <RETURN>'s to step through the example.

```
SPECIFY LINKER OPTIONS:
  (D) - DELETE UNSPECIFIED RSEGS
  (S) - INCLUDE SYMBOL TABLE
  (L) - LIST TABLES ON CRT
>LS
SPECIFY INPUT FILE
>INCR.R
RSEG      LENGTH
INCR       0006
SPECIFY INPUT FILE
>
SPECIFY OUTPUT FILE
>1:INCR
SPECIFY LISTING FILE
>
LINKER INPUT
>#END
0000   INCR       INCR.R      0006


FILE         RSEG        ADDR LENGTH
INCR.R       INCR        0000 0006

             GLOBALS
             INCR       0000


ENTRY POINT 0000
FUNCTION COMPLETED
```

Enter <RETURN> to continue with the module.

The next two displays are on the following page.

****
****

NOTES

To create a Z file (system file) the attribute must be changed from O to Z in the Manager. The program must start at O or there must be a jump to the start of the program at O for the reset key to be operational.

TO CONTINUE TYPE ^

****
****
****

NOTES

If system routines are being used two more RSEG's must also be used. They are WDATA and CRT (or DCRT).

TO CONTINUE TYPE ^

CRT must be linked on a 2K boundary. The location used by the system is X'D800'. WDATA must be linked at X'D600'.

****
****

This module is completed after the above two displays. The module will delete INCR and COUNTER from the disk in Drive 1 and then return to the menu.

If the module is aborted in the middle, the files INCR and COUNTER must be deleted from Drive 1 before the module will execute.

The menu is then displayed. To proceed, type the name of the feature you wish to see next.

****
****

GenRad/DSD

Advanced Development System

DEBUGGER

TO CONTINUE TYPE ^

Features:

Single step capability
Trace mode of execution
Real time emulation
Memory mapping in 256 byte blocks
Disassembled program flow
Software and hardware breakpoints

>

****
****

****
****

This segment demonstrates the use of the symbolic Debugger

to load, debug and execute assembly language programs in

GenRad/DSD

format

TO CONTINUE TYPE ^

JD

This command invokes the Debugger.

>
****
****

After typing ^, the module displays JD on the command line.
This invokes the Debugger.

To continue with this module enter ^.

The next page contains a sample Debugger display for the Z80.
The next 8 displays contain explanations of each area of the
Debugger display.

DEBUGGER DISPLAY

```
**********************************************************************************
**********************************************************************************
PC   A  B  C  D E  H L  IX   SZHVNC V   SP   A' B'C' D'E' H'L' IY   SZHVNC I
0000 00 00 00 0000 0000 0000 000000 00  0000 00 0000 0000 0000 0000 000000 00

0000 >C364D4      JP      X'D464'       FFB8  C0 C2 C0 C0  C0 80 C0 C0  ........
                                        FFC0  40 40 40 40  40 40 40 40  @@@@@@@@
0003  1D          DEC     E             FFC8  40 20 40 40  00 60 40 20  @ @@. '@
                                        FFD0  40 40 40 60  40 40 40 40  @@@'@@@@
0004  3D          DEC     A             FFD8  60 40 60 40  40 60 60 60  '@'@'@@@
                                        FFE0  40 00 40 40  40 40 40 40  @.@@@@@@
0005  1D          DEC     E             FFE8  00 40 40 60  40 40 00 00  .@@'@@..
                                        FFF0  40 40 40 40  40 40 40 40  @@@@@@@@
0006  3D          DEC     A             FFF8  60 20 60 00  40 00 60 00  ' '.@.'.
                                        0000 >C3 64 D4 1D  3D 1D 3D 1D  .d..=.=.
0007  1D          DEC     E             0008  2C 0C 1C 1D  2D 1D 0D 0D  ,...-...
                                        0010  9D 1D 5D 5C  5D BD 1D 1D  ..]/]...
0008  2C          INC     L             0018  1C 3D 0C 1C  2D 25 3D 1D  .=..-%=.
                                        0020  3D 1D 1D 1D  9D 3D 3D 3D  =....===
0009  0C          INC     C             0028  2C 04 1D 0D  0D 0D 2D 15  ,.....-.
                                        0030  C3 44 D4 2C  0D 1F 1D 1D  .D.,....
000A  1C          INC     E             0038  04 04 04 0C  0C 0E 1D 1D  ....$..%
                                        0040  18 0A 11 19  38 1B 19 19  ....9...
000B  1D          DEC     E             0048  19 19 19 1D  1B 19 1B 1D  ........
                                        0050  1A 18 18 1A  1B 1B 1B 1D  ........
>
**********************************************************************************
**********************************************************************************
```

Enter ^ to continue.

****
****

DEBUGGER DISPLAY

The display is divided into three areas:

TO CONTINUE TYPE ^

---

1. Register display on the top two lines
2. Disassembly display on the left side
3. Memory Map on the right side

>

****
****
****

REGISTER DISPLAY

The first line of the display shows the letters assigned by the manufacturer of the Z80 to its internal registers. The second line displays the contents of the registers and the program status byte in hexadecimal.

TO CONTINUE TYPE ^

The order of the registers for the Z80 is:

PROGRAM COUNTER
MAIN REG SET
INDEX REGISTER IX
INTERRUPT VECTOR
STACK POINTER
ALTERNATE REG SET
INDEX REGISTER IY
INSTRUCTION REGISTER

>

****
****

****
****

DISASSEMBLY DISPLAY

The left half of the display contains ten Z80 assembly language instructions. The first instruction is prefixed by >. This is a memory pointer indicating the next byte to be executed. The memory pointer address is also used as input by three other commands.

TO CONTINUE TYPE ^

Disassembly is performed as though memory contained a program. Since disassembly may start at any byte in memory, and memory locations may contain data rather than a program, the disassembly display may be meaningless. It is up to the user to position the left half of the display so that disassembly starts at the first byte of an instruction.

>

****
****
****

DISASSEMBLY DISPLAY

The first column on the left shows the hexadecimal memory suffixed with an * if it is write protected or a + if it is in the user system during emulation.

TO CONTINUE TYPE ^

The second column shows the hexadecimal display of the instruction. The third column shows the Z80 assembly language mnemonic corresponding to the hexadecimal equivalent in the second column. The fourth column shows the operand(s) which is (are) associated with the mnemonic in the third column.

>

****
****

****

MEMORY MAP

The right half of the Debugger display shows both an ASCII and a hexadecimal dump of 160 memory locations. A > points to the byte at which execution would begin. The memory pointer address is also used as input by three other commands.

TO CONTINUE TYPE ^

The first column on the right side of the display shows the hexadecimal address of the first of eight memory locations on each line. If a location in memory is write protected the address is suffixed with an *. If the memory location is placed in the user system during emulation, it is suffixed with a +.

>

****

MEMORY MAP

The next eight columns on the right side of the Debugger display are the contents of memory in hexadecimal. The first column is the contents of the displayed address. The addresses of the next 7 bytes can be obtained by counting.

TO CONTINUE TYPE ^

The last column is the ASCII representation of the contents of each of the locations in the memory map. This feature is useful if the contents of memory displayed are alphabetic data. If a byte does not contain a printable ASCII character a "." is displayed.

>

****

****

MEMORY POINTERS

As mentioned before there are two memory pointers, one on each side of the display, which point to the displayed contents of memory.

TO CONTINUE TYPE ^

These memory pointers indicate the byte at which execution would begin after an execute, trace or single step command. The memory pointer address is also used as input by three other commands.

>

****

COMMAND ENTRY CURSOR

On the command line (the bottom line) there are two additional pointers. The blinking cursor may be positioned in front of either pointer by depressing the <TAB> key. Commands affect the side on which the cursor is positioned. The side where the cursor is positioned is called the active side of the display.

TO CONTINUE TYPE ^

Upon entering the Debugger, only the pointer on the left side of the command line is displayed and both sides of the display are "slaved" together. The first time the <TAB> key is depressed, the pointer in the middle of the command line is displayed and commands will only affect the side where the cursor is located. The <RESET> key and a display command will "slave" the two sides together.

>

****

LOAD PROGRAM

L{filename}[,[offset][,symbol-table-address[,G]]]

This command loads absolute object programs so that they may be examined, changed, and executed using the Debugger.

TO CONTINUE TYPE ^

---

filename is the name of an absolute object file.

offset is a constant or expression specifying the number of of memory locations to be added to the load address given in the object file. If not specified, the offset value is zero. This parameter is used in programming EPROMs.

(explanation continued on next page)

>

This command loads a program from the diskette into the memory locations previously specified by the user during the Linker operation. The file must have an O attribute and be formatted properly.

The optional offset parameter allows the user to load the program into different memory locations than those specified to the Linker. For example, if a program in a file named TEST is loaded into memory starting at 100H but an offset of 50H is specified (LTEST,50), the program will be loaded into memory starting at location 150H. A program will not execute when a non-zero offset is specified since the addresses in the program are not altered when the program is loaded! The offset parameter is used primarily when programming PROMs. The commands for programming PROMs are explained starting on page 143.

To continue, enter ^.

****
****

LOAD PROGRAM

L[filename][,[offset][,symbol-table-address[,G]]]

This command sets the program counter to the entry point of the program specified by the Linker.

TO CONTINUE TYPE ^

---

symbol-table-address is a hex constant or an expression specifying the beginning address at which the program symbol table is to be loaded. Note that if this parameter is specified, but the offset parameter is not, the table parameter must be separated from filename by two commas. If symbol-table-address is not specified, the table will not be loaded.

>

****
****

The L command is the third and final step in using the Symbolic Debug features of the 2300 ADS. The S option in the Assembler is specified to include the symbol table in the relocatable code file and the S option in the Linker is specified to include the symbol the symbol table in the absolute code file. The L command loads the symbol table and the program into memory.

These three steps allow the user to access the labels of the assembly language program in debugging operations. The symbol-table-address parameter specifies the starting location in memory for loading the symbol table. The symbol table can be loaded into any unused RAM, but should not overlap the Debugger program or the user program.

When the symbol table is loaded, the length of the table is displayed on the line above the command line. The user can then determine if the symbol table was loaded into memory used by the system or by the user program.

The optional G paramater loads only the global symbols into memory, thus reducing the memory requirements of the symbol table.

To continue, enter ^.

****
****

LOAD PROGRAM

L{filename}[,[offset][,symbol-table-address[,G]]]

Three examples will follow next.

EXAMPLE NEXT, TO CONTINUE TYPE ^

Symbolic debugging requires that a symbol table be generated by the Assembler, output by the Linker, and loaded by the Debugger. When the symbol table is loaded, the Debugger displays its hexadecimal length on the second line from the bottom of the display.

>

****
****

After entering ^, the module jumps to the Debugger and displays the command "LINCR". When <RETURN> is entered, the system loads the file INCR. Notice that the PC is now 0300 which is the entry point for the program.

To continue, enter ^. The module clears the PC and enters the command "LINCR,100,2000". When <RETURN> is entered, the system loads the file INCR. Since the offset is 100, the program is loaded at location 400 enstead of location 300. Notice that the PC is again 0300 and the bottom line has the message "SYMTAB LENGTH=001D". The symbol table was loaded into memory starting at location 2000H and has a length of 1DH.

To continue, enter ^. The module clears the PC and enters the command "LINCR,,1000". This is the most common forms of the L command. When <RETURN> is entered, the system loads the program in INCR into memory and loads the symbol table into memory starting at location 1000. When <RETURN> is entered, the system sets PC=0300 and displays the message "SYMTAB LENGTH=001D". The user may now use this command.

To continue, enter ^.

****
****

SWAP COMMAND CURSOR

<TAB>

This command selects which half of the display will be affected by the next command.

TO CONTINUE TYPE ^

Upon entering the Debugger, both sides of the display are "slaved" together. The first time the <TAB> key is depressed, the two sides of the display are separated so commands will only affect the the side of the display where the cursor is located. The <RESET> key followed by a display command will "slave" the two sides together.

>

****
****

The <TAB> key allows the user to exercise the split screen feature of the Debugger display. The disassembled display is primarily used for displaying the next ten instructions to be executed. The memory map side of the display displays a data area which will be affected by the program on the disassembled side of the display. The <TAB> key allows the user to display dufferent areas of memory with the memory map display and the disassembled display.

To continue, enter ^.

****
****

DISPLAY MEMORY

D{expression}

This command displays the contents of memory specified by the expression.

EXAMPLE NEXT, TO CONTINUE TYPE ^

---

expression may be a constant, an arithmetic expression, an indirect address, a relative address, a symbolic reference or a combination of the above.

D$

This form of the command returns the previous display to the screen.

>

****
****

When ^ is entered, the module jumps to the Debugger, loads the program INCR into memory and displays the command "D100".

When <RETURN> is entered, both memory pointers are pointing to location 100 (both sides of the display are hooked together)> The module also displays the command "D#START". The # tells the Debugger that the character string immediately following will be found in the symbol table.

When <RETURN> is entered, the system displays location 300H, which is the start of the program. The module then displays the command "D#START+3". These examples illustrate the use of simple expressions in displaying memory locations.

When <RETURN> is entered, the system displays the memory location 3 bytes past the label START or location 303.

Enter <TAB> to show the split screen debug. Then enter ^ and the command "D1000 is displayed. Enter <RETURN> and the symbol table is displayed. In the ASCII dump it is possible to see the symbols used in the program.

To continue, enter ^.

****
****

POSITION MEMORY CURSOR

<RIGHT ARROW>

This command advances the memory pointer one byte on the side where the cursor is located. If the sides are tied together, both pointers are advanced one byte.

TO CONTINUE TYPE ^

<LEFT ARROW>

This command moves the memory pointer back one byte in memory on the active side of the display. If the sides are tied together, both pointers back up one byte. If the cursor is on the left side this will change the disassembled instructions which are displayed.

>

****
****

This display and the next display explain the effects of the cursor control keys on the area of memory displayed in the Debugger.

The left and right arrows can be used on the disassembled side of the display to align the memory pointer with an instruction boundary. It can also be used for positioning the memory pointer before using the Store Data in Memory command.

To continue, enter ^.

****
****

POSITION MEMORY CURSOR

<DOWN ARROW>

If the cursor is on the right side of the display or the sides of the display are tied together, the pointer(s) is(are) advanced by eight bytes. If the sides are separated and the cursor is on the left side, the pointer advances one instruction.

EXAMPLE NEXT, TO CONTINUE TYPE ^

<UP ARROW>

This command moves the appropriate pointer back eight bytes in memory under all conditions. It is not possible to disassemble backward, since intructions take up different numbers of bytes.

>

****
****

After entering ^, the module jumps to the Debugger and returns control of the system to the user. At this point the user may experiment with the cursor control keys and the <TAB> key for the split screen Debugger.

To continue, enter ^.

****
****

STORE DATA IN MEMORY

S{constant}[,constant]...

This command stores data entered at the keyboard directly into memory. A space may be used instead of a comma as a delimiter.

EXAMPLE NEXT, TO CONTINUE TYPE ^

---

constant is a one or two digit hexadecimal number or a single ASCII character surrounded by single quotes. If a single digit is used, the high order four bits stored will be zeroes.

The constants are stored beginning at the memory pointer address of the side where the cursor is located.

>

****
****

This command allows the user to make patches directly in the hex code in the Debugger. This command is also used to change the contents of a data area or set up the conditons for execution of a program.

When ^ is entered, the module jumps to the Debugger, displays 500H and enters the command "S3C D3 0 C3,0,01". This example illustrates that both commas and spaces may be used as delimiters between bytes and that the high order four bits are assumed to be zero if they are not specified.

When <RETURN> is entered, the Store command is entered and the display changes to reflect the new contents of memory. The next command "S41,'B' 'C',44" is also displayed. This demonstrates storage of ASCII characters in memory.

When <RETURN> is entered, the command is accepted and the display shows that the string ABCD is stored in memory starting at 500H.

The user may now try other examples of the Store Data command.

To continue, enter ^.

****
****

STORE DATA IN REGISTER

Z{reg=constant}[,reg=constant]...

This command stores data entered at the keyboard directly into a register. All registers may be changed except PC and I.

TO CONTINUE TYPE ^

If reg is an 8 bit register, constant may be one or two hex digits or a single ASCII character surrounded by single quotes.

8 bit registers

A B C D E H L F V A' B' C' D' E' F'

>

****
****

This command can be used to change the contents of the registers to set up conditions before executing a program. It may also be used along with breakpoints to simulate I/O when debugging software and before In-circuit Emulation.

To continue, enter ^.

****
****

STORE DATA IN REGISTER

Z{reg=constant}[,reg=constant]...

This command stores data entered at the keyboard directly into hardware registers. All registers may be changed except the PC and I register.

EXAMPLE NEXT, TO CONTINUE TYPE ^

If reg is a 16 bit register, constant may be a one to four digit hexadecimal constant or one or two ASCII characters surrounded by single quotes. If less than 4 hex digits or a single ASCII character are specified, the high order four bits are stored as zeros.

16 bit registers

X S Y

>

****
****

When ^ is entered, the module jumps to the Debugger and displays the command "ZA=3F". Notice that the A register now contains 00.

When <RETURN> is entered, the A register now contains 3F and the command "ZF=3" is displayed. This command changes the status of the flag register.

When <RETURN> is entered, the N and C flags are set and the next command "ZX='A'".

When <RETURN> is entered, the IX register now contains 0041 or the hex equivalent of A. The command "ZA'='A'" is displayed.

When <RETURN> is entered, the A' register contains the hex equivalent of an ASCII A. The next command is "ZS=4567".

When <RETURN> is entered, the stack pointer will contain 4567 instead of 6E00. The user is now free to use the Z command.

To continue, enter ^.

****

FIND DATA IN MEMORY

F[constant][,constant]...

This command searches memory, starting at the memory pointer on the active side of the display, for a match between the constants specified and a byte or series of bytes in memory. When a match is found the pointer is positioned at the first byte of the match.

EXAMPLE NEXT, TO CONTINUE TYPE ^

---

constant is a one or two digit hex number or a single ASCII character surrounded by single quotes. If a single digit or a single ASCII character is used, the high order four bits will be zeroes. Unless a match is found, the search wraps around memory and stops at the memory pointer address.

If no parameters are specified, the previous constant or constants specified in an F command is (are) used.

>

****

When ^ is entered, the module jumps to the Debugger and displays the command "F41,'B' 'C',44".

When <RETURN> is entered, the system will find the first occurence of the string ABCD in memory starting from location zero. The memory pointers display location 100H. On the dissambled side of the display the first four lines ar 41, 42, 43, and 44. This same pattern can be seen on the memory map side in both the hex and ASCII dump. The module also shows the command F.

When <RETURN> is entered, the system accepts the F command. This finds the next occurence of the pattern in the last Find command. The display then shows location 500H.

The user may then use the Find command to display the location of another or the same pattern in memory.

To continue, enter ^.

BREAKPOINTS

A breakpoint is a location in a program at which point execution is stopped and control is returned to the Debugger. Breakpoints are useful for splitting a program into pieces which may then be executed separately. This makes debugging a complex program more manageable.

TO CONTINUE TYPE ^

There are three types of breakpoints:

1 hardware breakpoint in the Debugger.
4 software breakpoints in the Debugger.
3 hardware breakpoints in the Logic Analyzer.

(Logic Analyzer breakpoints explained in Logic Analyzer module.)

The next three pages and the next six displays explain the use of the Debugger breakpoints. Examples of the breakpoints are shown in conjunction with examples of the modes of execution.

To continue, enter ^.

****

DEBUGGER HARDWARE BREAKPOINT

The hardware breakpoint in the Debugger is referred to as breakpoint 0. This breakpoint monitors the address lines and causes a hardware interrupt which returns control to the Debugger when the address set as a breakpoint is referenced.

TO CONTINUE TYPE ^

The hardware breakpoint will stop execution in the E mode or in the T mode. It is possible however, to single step through the hardware breakpoint.

>

****

SOFTWARE BREAKPOINTS

The four software breakpoints are numbered from 1 to 4. When a software breakpoint is set, an F7 is stored at the memory location of the breakpoint. This is a Z80 interrupt instruction. The instruction in the memory location occupied by the software breakpoint is automatically restored when the breakpoint is cleared.

TO CONTINUE TYPE ^

The software breakpoint stops execution in the E mode, the T mode and in the single step mode. (It is not possible to step through a a software breakpoint.)

>

****

BREAKPOINT SET COMMAND

BS<n>

This command sets a breakpoint at the address of the memory pointer.

TO CONTINUE TYPE ^

---

n is an integer between 0 and 4.

If n=0, the hardware breakpoint is set. If n is between 1 and 4 inclusive, a software breakpoint is set.

BREAKPOINT DISPLAY COMMAND

BD<n>

This command positions the memory pointer at the address of the nth breakpoint.

TO CONTINUE TYPE ^

---

n is an integer between 0 and 4.

If n=0, the hardware breakpoint is be displayed. If n is between 1 and 4 inclusive, a software breakpoint is displayed.

****

****

BREAKPOINT RESET COMMAND

BR

This command removes the breakpoint at the address indicated by the memory pointer. If there is no breakpoint at that address, the message NOT ACTIVE is displayed and no action is taken.

TO CONTINUE TYPE ^

---

n is an integer between 0 and 4.

If n=0, the hardware breakpoint is reset. If n is between 1 and 4 inclusive, the software breakpoint is removed and the former instruction is restored in memory.

>

****

****

****

BREAKPOINT CLEAR COMMAND

BC

This command removes all breakpoints but does not restore the former instructions in memory.

TO CONTINUE TYPE ^

>

****

****

****

EXECUTE SINGLE INSTRUCTION (SINGLE STEP)

<STEP> key

A single instruction at the address of the memory pointer is executed.

EXAMPLE NEXT, TO CONTINUE TYPE ^

Upon completion of the instruction, the memory pointer is moved to the address indicated by the program counter, i.e. the address of the next instruction to be executed. If the executed instruction was a jump or a call, the display may show a totally different area of memory than before execution. D$ will return to the former display.

****

The single step is the first of three modes of execution. It is usually the first step in debugging the software.

After entering ^, the module jumps to the Debugger, loads the program INCR and display the start of the program. The memory pointers should be displaying location 300H. At this point the user may press the <STEP> key once. Notice that the instruction was an INC A and that the A register now contains 01. The PC is now 0301. The user is free to step through the program.

After entering another ^, the module will load the program counter and display its entry point. The user is once again free to exercise the <STEP> key and any of the other Debugger functions discussed above.

To continue, enter ^.

****

TRACE EXECUTION

T[expression]

Trace execution allows the user to save the contents of the program counter, registers and status bytes after the execution of each of 20 instructions prior to a breakpoint.

TO CONTINUE TYPE ^

Trace execution starts at the address specified by the value of the expression. If the expression is omitted, trace execution is begun at the memory pointer address of the active side of the display.

T$

Trace execution starts at the address specified by the contents of the program counter.

>

****

Examples of Trace mode of execution are shown after the next display.

To continue, enter a ^.

****
****

TRACE EXECUTION

This command interrupts execution after every instruction to save the contents of all the registers. Twenty of these traces are saved, with new traces added at the bottom of a stack in memory and old ones removed from the top. When a breakpoint is encountered, the last 20 traces are displayed.

EXAMPLE NEXT, TO CONTINUE TYPE ^

The display is returned to the Debugger by typing any key on the keyboard.

TR

This command displays the data saved in a T (Trace Execution) command. The last trace is at the bottom of the display.

****
****

After entering ^, the module will jumps to the Debugger, loads the program counter, displays the entry point of counter and shows the command "D#TIME1+5".

When <RETURN> is entered, a jump relative (JR) is displayed. The command BS1 is also displayed on the command line.

When <RETURN> is entered, software breakpoint #1 is placed at location 30B. Notice that the code is changed and the instruction RST X'30' is now displayed. This is the software breakpoint. The command "T#COUNTO" is also displayed.

When <RETURN> is entered, the Debugger will start tracing from the location of the label COUNTO and execute until the breakpoint is encountered. The screen momentarily blanks out and then the trace is displayed. Notice that the last instruction at the bottom of the screen is the software breakpoint.

When <RETURN> is entered, the Debugger displays returns and the command BR.

When <RETURN> is entered, breakpoint #1 is removed and the TR command is displayed.

When <RETURN> is entered, the trace display returns.

To continue, <RETURN> to the Debugger display and enter ^.

****
****

EXECUTE PROGRAM

E[expression]

Execution starts at the address specified by the value of the expression with maskable interrupts enabled. If the expression is omitted, execution is begun at the address of the memory pointer on the active side of the display.

TO CONTINUE TYPE ^

E*

Execution starts at the address specified in the program counter.

This execution is done at 4 Mhz with no wait states. The display goes blank during execution.

>

****
****

To continue, enter ^.

NO-INTERRUPT EXECUTION COMMAND

N[expression]

Execution starts at the address specified by the value of the expression with maskable interrupts disabled. If the expression is omitted, execution is begun at the address of the memory pointer on the active side of the display.

TO CONTINUE TYPE ^

N$

Execution starts at the address specified in the program counter.

This execution is done at 4 Mhz with no wait states. The display goes blank during execution. This form of execution is very useful when debugging interrupt systems and software.

>

When ^ is entered, the module jumps to the Debugger, loads the program INCR and displays its entry point. The user is now free to set breakpoints and use the E or N command. This program is an infinite loop; if no breakpoints are set in the loop, execution will continue until the <BREAK> key is typed.

To continue, enter ^.

WRITE PROTECT MEMORY

X

Switches protection on or off for the 1024 (decimal) block of memory in which the memory pointer of the active side of the display is located.

TO CONTINUE TYPE ^

This command prevents writing to memory. Write protection assures that a faulty program will not write into memory locations which contain program or data meant only to be read.

>

After entering ^, the module will jump to the Debugger and display the X command.

When <RETURN> is entered, the 1K block starting from 0 will be write protected. Notice that a * appears next to the addresses which are write protected. Another X command is also displayed.

When <RETURN> is entered, the write protection is turned off for the 1K block starting from 0. Notice that the *'s disappear from the display. The user is now free to use the Write Protect command.

To continue, enter ^.

WRITE PROGRAM TO FILE

W{file-name}{,start-addr,end-addr}[,start-addr,end-addr]...[,entry-addr]

Any series of memory locations may be written to a diskette file as absolute binary using this command. The contents of memory are not affected by this command.

TO CONTINUE TYPE ^

All addresses are expressions or hexadecimal constants. The specified memory addresses are written to the file specified in absolute object format. If entry-addr is omitted, the first start-addr is used as the entry point address.

This command is used most often to save code which has been patched in the Debugger before returning to the Editor to change the source. Another common use for the command is to write the information read from a PROM to the disk or to write the program from the prototype's memory to the disk.

To continue, enter ^.

****

EPROM PROGRAMMING

The GenRad 2300 ADS EPROM programmer programs four different UV-erasable memories: 2704, 2708, 2758, 2716 and 2732. EPROM programming commands are available during the operation of the Debugger.

TO CONTINUE TYPE ^

The GenRad 2300 ADS also has software which will allow it to interface (through serial port 1) with the PROLOG 900B and the PROLOG 920. Software is also available through the user's group to interface with many other universal PROM programmers.

>

****

The following displays and commands are used with the EPROM personality modules for the 2704, the 2708, the 2716, the 2758 and the 2732.

To continue, enter ^.

CHECK FOR EPROM EMPTY

C[type][,length[,interleaving factor]]

If the EPROM being checked is empty, the message EPROM IS EMPTY is displayed. Otherwise the Debugger gives the address of the first location which is not empty.

TO CONTINUE TYPE ^

---

type is the EPROM type. It may be 2704, 2708, 2758, 2716 or 2732. If type is omitted, 2708 is assumed.

length is the number of bytes to be checked. If omitted, the entire EPROM is checked.

interleaving factor is either 1 or 2. It defaults to 1.

PROGRAM EPROM

P[type][,length[,interleaving factor]]

Data from the 2300 memory will be transferred beginning at the memory pointer address of the active side of the display. The EPROM is checked after it has been programmed.

TO CONTINUE TYPE ^

---

type is the EPROM type. It may be 2704, 2708, 2758, 2716 or 2732. If type is omitted, 2708 is assumed.

length is the number of bytes to be programmed. If omitted the entire EPROM is programmed.

interleaving factor is either 1 or 2. It defaults to 1.

****

INPUT DATA FROM EPROM

I[type][,length[,interleaving factor]]

EPROM data is stored in ADS memory beginning at the memory pointer address of the active side of the display.

TO CONTINUE TYPE ^

---

type is the EPROM type. It may be 2704, 2708, 2758, 2716 or 2732. If type is omitted, 2708 is assumed.

length is the number of bytes to be input in the ADS. If omitted, the entire EPROM is input.

interleaving factor is either 1 or 2. It defaults to 1.

>

****

VERIFY DATA IN EPROM

V[type][,length[,interleaving factor]]

Data in the EPROM is verified against data in ADS memory beginning at the memory pointer address of the active side of the display. If the verification fails, the address of first mismatch is displayed.

TO CONTINUE TYPE ^

---

type is the EPROM type. It may be 2704, 2708, 2758, 2716 or 2732. If type is omitted, 2708 is assumed.

length is the number of bytes in the EPROM to verify. If omitted, the entire EPROM is verified.

interleaving factor is either 1 or 2. It defaults to 1.

>

****

This completes the Split Screen Symbolic Debugger Module. To go on to the next module, enter ^. The menu will be displayed. Type the name of the feature you wish to see next.

GenRad/DSD

Advanced Development System

Z80 IN-CIRCUIT EMULATOR

TO CONTINUE TYPE ^

This segment explains the use of the Z80 In-Circuit Emulator to verify the operation of user hardware.

To continue, enter ^.

In-Circuit Emulator

The In-Circuit Emulator is a hardware feature that enables the microprocessor in the GenRad 2300 ADS to emulate the microprocessor in the user system. A connector (Emulator Probe) on the end of the end of the emulator cable is plugged directly into the microprocessor socket on the user's prototype.

TO CONTINUE TYPE ^

The emulation package consists of the following elements:

1. An In-Circuit Emulator interface card.
2. Cable to connect the interface card to the CPU card.
3. An In-Circuit Emulator Probe assembly.
4. Logic Analyzer intertface card.

>

In-Circuit Emulator

The In-Circuit Emulator can be used to substitute RAM from the 2300 ADS for the PROM/ROM in the user system. The RAM can be much more easily modified than the ROM/PROM thus facilitating changes in user programs during the development phase. Emulation also provides debugging facilities in the user environment.

TO CONTINUE TYPE ^

Features such as single-step, trace, and breakpoints are all included in the emulation package. This package allows the user to gradually bring up the target system. The clock, DMA facilities, interrupts and I/O and memory mapping may all be separately enabled.

>

To continue, enter ^.

****

In-Circuit Emulator

Initially the 2300 ADS is used to debug and develop the user software. Then the emulator is used to verify the operation of user hardware. The designer may switch clocks, control lines and memory from the 2300 to the target system. As various elements of the target system are verified, more and more of these elements are utilized until the emulation connector is removed.

TO CONTINUE TYPE ^

This step by step procedure is extremely effective since only one element at a time is being tested. All of the 2300 Debugger commands are available to the user during emulator operation. Thus the user is able to examine and alter memory and registers, start a program, set breakpoints, single-step, trace, load and dump object programs.

****

Z80 EMULATOR FEATURES

1. Real time emulation and debugging at 4Mhz clock cycles using high-speed static RAM.

2. Separate control over clock source, DMA, Interrupt and I/O, memory map and refresh enable.

TO CONTINUE TYPE ^

---

3. Memory mapping in 256 byte blocks under software control.

4. Single step, trace and 4 software breakpoints.

****

To continue, enter ^.

****

Z80 EMULATOR FEATURES

5. Logic Analyzer adds:
   -- 3 complex breakpoints
      16 address, 8 data, Memory/IO, R/W, Instruction fetch, DMA cycle and 4 external conditions.
   -- trace qualifer
   -- 256 deep trace memory

TO CONTINUE TYPE ^

---

6. The <BREAK> key, Logic Analyzer breakpoints, single-step and trace all use the interrupt vector at X'66' (NMI). The software breakpoint uses the interrupt vector at X'30' (RESTART 6). The <RESET> key uses the interrupt vector at X'0' (RESTART 0).

>

****

Z80 EMULATOR FEATURES

7. A system with 48K of RAM provides 27K of ROM/PROM simulator memory from X'0000' to X'6DFF'. A system with 64K of RAM provides 43K of ROM/PROM simulator memory from X'0000' to X'6DFF', from X'A000' to X'BFFF' and from X'E000' to X'FFFF'.

TO CONTINUE TYPE ^

---

8. The user may address memory in the target system from X'0000' to X'FFFF' except for a 256 byte Debugger communication block resident in the emulator at X'D400' to X'D4FF'. This communication block is relocatable.

9. All I/O ports are available to the user except for output port X'F6'.

>

****

To continue, enter ^.

EMULATOR MODE COMMAND

M[mode][mode]...

mode may be any combination of the following:

C Enables the user system to clock line input. This mode causes the Emulator to switch from the ADS internal clock to the target system clock.

TO CONTINUE TYPE ^

D Allows the target system to generate DMA requests which suspend of microprocessor operation. This allows user DMA logic to gain control of the bus and generate transactions to target system memory (Note: the user DMA logic cannot access ROM/PROM simulator memory.) For the Z80 the DMA request is caused by a low level (<0.4 VDC) on the BUSRQ- line and the DMA grant signal is indicated by a low level on the BUSAK- line.

The emulator mode command (M) allows the user to determine which resources are utilized in the prototype and which resources remain in the ADS.

The C mode allows the user to test the clock on the prototype. If the 2300 continues to operate, the clock in the prototype is functioning. If the 2300 screen goes blank, the clock on the prototype is inoperative.

The D mode allows the user to test a prototype which has more than one microprocessor on it. Without the D option, all DMA requests from other microprocessors on the prototype (other than the one which is being emulated) are ignored. Thus the software may be debugged without the interference of the other microprocessors. When the D option is specified, the user may then debug the system with the interactions between processors on the prototype.

To continue, enter ^.

****
****

EMULATOR MODE COMMAND

I Allows the target system to control the following input lines: RESET-, INT-, WAIT- and NMI. The normal (inactive) conditions for these lines are:

RESET- = high (>2.4 VDC)
INT- = high
WAIT- = high
NMI- = high

TO CONTINUE TYPE ^

Note especially that a low on the WAIT- line will completely halt the microprocessor. The I mode parameter also enables the emulator to direct I/O requests to the target system. For the Z80 emulator the user must be aware of the emulator control port (output port X'F6').

>

****
****

The I mode allows the user to access the I/O space in the prototype. Until the I option is specified, the user must use breakpoints and the Zap Register (Z) command to simulate the input instructions.

To continue, enter ^.

EMULATOR MODE COMMAND

M Enables the memory mapping function. See the USERMAP command for details.

E The E mode is a combination of C, I, M, and D.

TO CONTINUE TYPE ^

R The R mode disables the CRT and Memory refresh during user program execution. Of course it is not possible to use this mode parameter without the static RAM option, since disrupting the refresh will cause both the user program and the Debugger to be lost if they are stored in dynamic RAM.

If the R option is not selected, the system is running at about 86% of full speed. The 14% is stolen in DMA cycles to refresh the RAM and the CRT.

For real time emulation, the user should specify both the E and the R emulator modes (MER).

The following pages and displays give additional infomation about the In-Circuit Emulator features.

To continue, enter ^.

****

EMULATOR MODE COMMAND

If the M command is entered without any mode parameters, the system returns to full host mode and all inputs, the clock, memory and I/O requests are be directed to the host system.

TO CONTINUE TYPE ^

The user should be aware of the exact timing of the various, mode switches. The C and D mode parameters will cause an immediate switch of the affected control lines as soon as the <RETURN> key is typed following MC, MD, MCD etc. The R mode parameter does not switch off the refresh function until the user program is executed. The refresh function is switched back on as soon as control is returned to the Debugger after a break from the user program.

>

****

EMULATOR MODE COMMAND

The I mode parameter enables the interrupt, reset, ready control lines and I/O requests when the user program is executed and disables them when control is returned to the Debugger.

TO CONTINUE TYPE ^

The M mode parameter enables the Memory Mapping function whenever a memory access is made (for example to display memory, alter memory, execute a user program etc.).

>

****

****
****

USER MEMORY MAP COMMAND

U[address],{mask}

This command is used to specify what part of ADS memory
is mapped to the prototype when using the Emulator Mode Command (MM).

TO CONTINUE TYPE ^

---

address must be a 2K boundary.

0000	4000	8000	C000
0800	4800	8800	C800
1000	5000	9000	D000
1800	5800	9800	D800
2000	6000	A000	E000
2800	6800	A800	E800
3000	7000	B000	F000
3800	7800	B800	F800

****
****

The U command allows the user to specify which areas of the memory space will be in the prototype during emulation. To enable the map the M mode in the emulator mode command must be selected.

After using the U command a "+" appears next to those addresses which will be in the prototype during emulation.

To continue, enter ^.

****
****

USER MEMORY MAP COMMAND

U[address],{mask}

mask is two hex digits. Each bit in the mask will switches 256 bytes into the user system. The bits are individually "OR'ed" to form the mask. The mask starts at the address specified in in the command. If no address is specified, map begins at the position of the memory pointer.

TO CONTINUE TYPE ^

The lowest bit in the mask corresponds to the lowest 256 byte block starting at the 2K boundary. The next higher bit corresponds to the next higher 256 byte block. All addresses may be mapped into the user system except the Debugger communication block which resides at X'D400' in the standard Logic Analyzer Debugger and emulator control port X'F6'. Whenever a memory location is acessed that has been mapped into the user system, the memory request and read or write is directed to the user system instead of the ADS ROM/PROM simulator memory.

>

****
****

This is the end of the Emulator module. For further information, consult the Emulator Reference Manual.

To return to the menu, enter ^.

****
****

GenRad/DSD

Advanced Development System

UNIVERSAL LOGIC ANALYZER

TO CONTINUE TYPE ^

Features:

View data bus during execution
256 traces
48 bit wide memory
Information recorded synchronously
with system or prototype clock
to 4 Mhz

****
****

****
****

This segment demonstrates the use of the Logic Analyzer

in troubleshooting and debugging tasks to view bus data

during program execution.

TO CONTINUE TYPE ^

The Logic Analyzer gathers a maximum of 256 traces for 16 address lines, 8 data lines, the READ/WRITE line, the I/O line, the DMA line, the Instruction Fetch line, and four external user-defined lines. Information is recorded synchronously with the 4Mhz clock for the Z80 either in the system or in the prototype.

>

****
****

The Universal Logic Analyzer for the Z80 monitors sixteen address lines, eight data lines, the I/O line, the RW line, the DMA line, the Instruction Fetch line (M1) and four external lines which may be selected by the user and monitored through the Logic Analyzer probes.

The Logic Analyzer is a synchronous logic analyzer and thus only stores what the microprocessor sees. This allows the user to verify that the lines to the microprocessor are either high or low depending upon the circumstances.

The Logic Analyzer hardware remains exactly the same for the 8080, the 8085 and the Z80. The Logic Analyzer requires only a different interface card to work in a 6800 or a 6802 system.

To continue, enter ^.

****
****

LOGIC ANALYZER ENTRY

A

From the Debugger display, this command enters the Logic Analyzer display.

TO CONTINUE TYPE ^

****
****

To access the Logic Analyzer requires not only the Logic Analyzer hardware but also the Logic Analyzer Debugger. This Debugger is called DEBUGZ80L on the RDOS disk. The Debugger called DEBUGZ80D does not contain the Logic Analyzer code or the prom programming code.

When ^ is entered, the module jumps to the Debugger and displays the A command.

To get to the Logic Analyzer from the Logic Analyzer Debugger, (DEBUGZ80L) the user enters an A<RETURN>. If this command is entered in DEBUGZ80D, a SYNTAX ERROR message is displayed.

When <RETURN> is entered, the Logic Analyzer display is shown on the CRT.

To continue, enter ^.

****

LOGIC ANALYZER EXIT

D

Returns to the Debugger address previously displayed.

D$

Returns to the Debugger with the display from the previous D command.

TO CONTINUE TYPE ^

D{expression}

Returns to the Debugger at the location of the expression. The expression may consist of a constant, an indirect address a relative address, a symbolic address, or a combination of such values.

>

****

When ^ is entered, the module jumps to the Debugger and displays the A command. Notice that the memory pointer is located at location 100.

When <RETURN> is entered the Logic Analyzer display is shown. The D command is also displayed.

When <RETURN> is entered, the display returns to the Debugger at location 100 (the display before entering the Logic Analyzer) and an A command is displayed.

When <RETURN> is entered, the display returns to the Logic Analyzer and the "D200" command is displayed.

When <RETURN> is entered, the display returns to the Debugger at location 200 and the A command is displayed.

When <RETURN> is entered, the display returns to the Logic Analyzer and the D$ command is entered.

When <RETURN> is entered, the display returns to the Debugger at location 100 and the A command is displayed.

When <RETURN> is entered, the display returns to the Logic Analyzer. The user may now try the command for entering the Debugger and Logic Analyzer. To continue, enter ^.

****

LOGIC ANALYZER DISPLAY

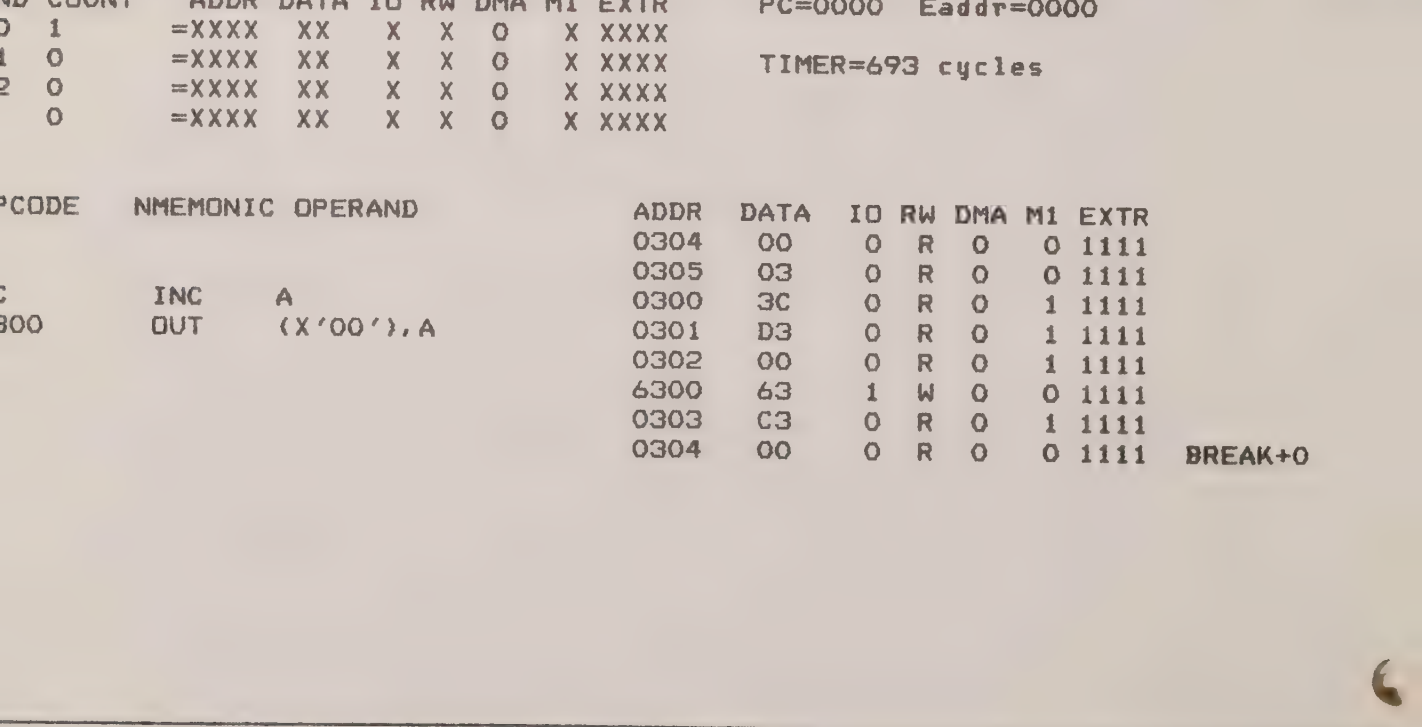
Upon entry to the Logic Analyzer, the four major parameters are displayed in the top third of the screen. These parameters are the AND/OR provision, the three hardware breakpoints, the Trace Qualifier and the timer increment. These parameters are explained below.

TO CONTINUE TYPE ^

****

****

LOGIC ANALYZER DISPLAY



```
ND COUNT    ADDR DATA IO RW DMA M1 EXTR      PC=0000  Eaddr=0000
0  1        =XXXX  XX   X  X  0   X XXXX
1  0        =XXXX  XX   X  X  0   X XXXX     TIMER=693 cycles
2  0        =XXXX  XX   X  X  0   X XXXX
   0        =XXXX  XX   X  X  0   X XXXX

PCODE   NMEMONIC OPERAND                ADDR  DATA  IO RW DMA M1 EXTR
                                        0304  00     0 R  0   0 1111
                                        0305  03     0 R  0   0 1111
C         INC    A                      0300  3C     0 R  0   1 1111
300       OUT    (X'00'),A              0301  D3     0 R  0   1 1111
                                        0302  00     0 R  0   1 1111
                                        6300  63     1 W  0   0 1111
                                        0303  C3     0 R  0   1 1111
                                        0304  00     0 R  0   0 1111  BREAK+0
```

****

AND/OR PROVISION

O

The OR provision causes the Logic Analyzer to halt the storage process when any one of the three breakpoint conditions has been met.

TO CONTINUE TYPE ^

A

The AND mode of operation requires that the designated breakpoint conditions be satisfied sequentially before the Logic Analyzer will stop data storage. First B0 conditions must be satisfied, then B1 conditions, then B2 conditions. It is not possible for B1 conditions to be satisfied before B0 has been satisfied.

>

****

When ^ is entered, the module jumps to the Debugger, enters the Logic Analyzer and displays the O command.

When <RETURN> is entered, the breakpoints are now OR'ed together instead of being AND'ed together. The A command is also entered.

When <RETURN> is entered, the breakpoints are again AND'ed together.

To continue, enter ^.

****
****

LOGIC ANALYZER BREAKPOINTS

The Logic Analyzer provides three hardware breakpoints labelled B0, B1, and B2. The Logic Analyzer stops the storage process when the specified conditions have been met.

TO CONTINUE TYPE ^

BX[,parameter=condition][,prarameter=condition]...

X is a breakpoint number (0, 1 or 2).
parameter and conditions are in table below.
If no parameters are specified, the breakpoint is reset to its default conditions.

****
****

The Logic Analyzer breakpoints set by the user determine the conditions that stop the Logic Analyzer from storing information on program execution.

These breakpoints can be used to break on the fourth I/O write to port X'5C', bracket a program so a break occurs whenever the program jumps out of the memory space allocated to it, or many other conditions.

The next six displays and the next three pages explain how to set the breakpoint conditions and their effects.

To continue, enter ^.

BREAKPOINT PARAMETERS

Parameter	Condition	Explanation
COUNT (or C)	=0	Breakpoint immediately satisfied.
	=1 to 65535 (decimal)	Number of times breakpoint conditions must be satisfied before breakpoint is satisfied.
ADDR (or A)	=1 to 4 hex characters	Missing characters are assumed to be high order zero hex digits.
	=#SYMBOL	Address from symbol table corresponding to SYMBOL is stored.

continued

>

BREAKPOINT PARAMETERS

Parameter	Condition	Explanation
ADDR (or A)	=XXXX	Don't care. Hex characters and X can be mixed, e.g.: ABCD X123 X1X1
	={expression}	Break when ADDR equals any allowable address expression.
	>={expression}	Break when ADDR is greater than or equal to {expression}. "X" is not allowed as part of the expression.

continued

>

BREAKPOINT PARAMETERS

Parameter	Condition	Explanation
ADDR (or A)	<=<expression>	Break when ADDR is less than or equal to <expression>. "X" is not allowed as part of the expression.
DATA (or D)	=1 or 2 hex characters	A missing character is assumed to be a high order zero.
	=XX	Don't care. Hex characters and X can be mixed, e.g.: AB AX X1

continued

BREAKPOINT PARAMETERS

Parameter	Condition	Explanation
IO	=1	Break on I/O execution.
	=0	Break on other than I/O execution.
	=X	Don't care.
RW	=R	Break on READ cycle. (memory or I/O)
	=W	Break on WRITE cycle. (memory or I/O)
	=X	Don't care.

continued

BREAKPOINT PARAMETERS

Parameter	Condition	Explanation
DMA	=1	Break on DMA cycle.
	=0	Break on other than DMA cycle.
	=X	Don't care.
M1	=1	Break if line high.
	=0	Break if line low.
	=X	Don't care.

continued

BREAKPOINT PARAMETERS

Parameter	Condition	Explanation
EXTR (or E)	=4 binary digits	Must have 4 binary digits.
	=1	Break if line high.
	=0	Break if line low.
	=X	Don't care.

After entering ^, the module jumps to the Debugger, goes to the Logic Analyzer and displays the command "B0,A=#START, C=65000,M1=1". After this command is entered, breakpoint 0 will be satisfied on the 65000th instruction fetch from the memory address START. This command demonstrates that the symbolic debugging features of the Debugger are also applicable in the Logic Analyzer.

After <RETURN> is entered, the appropriate parameters for B0 are set. The command "B1,C=1,A=XX2F,IO=1,RW=W" is also displayed. This command causes breakpoint 1 to be satisfied the first time an IO write is performed to port X'2F'.

After <RETURN> is entered, B1 is set according to the last instruction. The instruction B0 is also displayed.

After <RETURN> is entered, the conditions for breakpoint 0 return to their default conditions. The instruction B1 is also displayed.

After <RETURN> is entered, the conditions for breakpoint 1 return to their default conditions. The next instruction, "B0,A<=#START-1,M1=1", is displayed.

After <RETURN> is entered, the conditions for breakpoint 0 show that it will be satisfied by the first instruction fetch to an address less than 300H. "B1,C=10,A>=306,M1=1" is the next instruction displayed.

After <RETURN> is entered, the conditions for breakpoint 1 show that it will be satisfied by the tenth instruction fetch to an address greater than 306H. The first nine instructions are Debugger instructions which are executed after the Logic Analyzer is turned on but before the user program is executed. These instructions are located in the 256 byte communication block which is located from X'D400' to X'D4FF' in the standard Debuggers. "B2,C=1,A<=2FF,M1=1" is the next intruction displayed.

After <RETURN> is entered, the conditions for B2 will be the same as for breakpoint 0. This is done so that B2 will not be satisfied before B0 or B1 when the breakpoints are OR'ed together. An 0 instruction is then displayed.

After <RETURN> is entered, the breakpoints are OR'ed together. This results in a break the first time an instruction is fetched from a location other than between 300 and 305. This brackets an area of memory and is an effective way to isolate unexplained jumps out of the program area.

To continue, enter ^.

****
****

LOGIC ANALYZER TRACE QUALIFIER

Q[,parameter=condition][,parameter=condition]...

The Trace Qualifier specifies the characteristics of the events to be stored in the Logic Analyzer memory. The count parameter may be entered to continue storing for a specified number of bus cycles after the breakpoint conditions have been satisfied.

TO CONTINUE TYPE ^

Q with no parameters will reset the Qualifier to its default condition.

Trace Qualifier parameters are detailed below.

>

****
****

The next three pages and six displays explain the effects of the Qualifier register parameters.

To continue, enter ^.

TRACE QUALIFIER PARAMETERS

Parameter	Condition	Explanation
COUNT (or C)	=0	No qualification of stored data.
	=1 to 65535 (decimal)	Number of qualified cycles to be stored after breakpoint conditions are satisfied (DELAY).
ADDR (or A)	=1 to 4 hex characters	Missing characters are assumed to be high order zero hex digits.
	=#SYMBOL	Address from symbol table corresponding to SYMBOL is stored.

continued

TRACE QUALIFIER PARAMETERS

Parameter	Condition	Explanation
ADDR (or A)	=XXXX	Don't care. Hex characters and X can be mixed, e.g.: ABCD X123 X1X1
	={expression}	Store when ADDR equals allowable address expression.
	>={expression}	Store when ADDR is greater than or equal to {expression}. "X" is not allowed as part of the expression.

continued

TRACE QUALIFIER PARAMETERS

Parameter	Condition	Explanation
ADDR (or A)	<={expression}	Store when ADDR is less than or equal to {expression}. "X" is not allowed as part of the expression.
DATA (or D)	=1 or 2 hex characters	A missing character is assumed to be a high order zero.
	=XX	Don't care. Hex characters and X can be mixed, e.g.: AB AX X1

continued

TRACE QUALIFIER PARAMETERS

Parameter	Condition	Explanation
IO	=1	Store on I/O execution.
	=0	Store on other than I/O execution.
	=X	Don't care.
RW	=R	Store on READ cycle. (memory or I/O)
	=W	Store on WRITE cycle. (memory or I/O)
	=X	Don't care.

continued

TRACE QUALIFIER PARAMETERS

Parameter	Condition	Explanation
DMA	=1	Store on DMA cycle.
	=0	Store on other than DMA cycle.
	=X	Don't care.
M1	=1	Store if line high.
	=0	Store if line low.
	=X	Don't care.

continued

TRACE QUALIFIER PARAMETERS

Parameter	Condition	Explanation
EXTR (or E)	=4 binary digits	Must have 4 binary digits.
	=1	Store if line high.
	=0	Store if line low.
	=X	Don't care.

LOGIC ANALYZER QUALIFIER REGISTER EXAMPLES

After entering ^, the module jumps to the Debugger, goes to the Logic Analyzer display and displays the command "G,C=10".

When <RETURN> is entered, this command causes the Logic Analyzer to continue storing information for ten additional M cycles after the breakpoints have been satisfied. The next command, G is also displayed.

When <RETURN> is entered, Qualifier register parameters restored to their default conditions. The command "G,A=XX3D,IO=1" is displayed.

When <RETURN> is entered, the Logic Analyzer now stores only IO operations to port X'3D'.

To continue, enter ^.

****
****

LOGIC ANALYZER TIMER

The Logic Analyzer has a timer using a 10Mhz clock on the Logic Analyzer boards which measures the length of time between the satisfaction of breakpoints B0 and B1. It measures the time in either bus cycles, microseconds or 100 nanoseconds.

TO CONTINUE TYPE ^

TB
Set the timer increment to bus cycles.
TU
Set the timer increment to microseconds.
TN
Set the timer increment to 100 nanoseconds.

****
****

When the T command is used, only the unit of time is changed, not the contents of the timer register. Thus the next time the program is executed the timer will be timing in the new units.

When ^ is entered, the module jumps to the Debugger, enters the Logic Analyzer and displays the command TN.

When <RETURN> is entered, the unit of measure for the timer is changed from bus cycles to nanoseconds. The command TU is also displayed.

When <RETURN> is entered, the unit of measure for the timer is changed from nanoseconds to microseconds. The command TB is also displayed.

When <RETURN> is entered, the unit of measure for the timer is changed from microseconds to bus cycles.

To continue, enter ^.

The next two displays show the Logic Analyzer default conditions and explain the top part of the Logic Analyzer display.

```
**********************************************************************************
**********************************************************************************

                          DEFAULT CONDITIONS

  AND  COUNT    ADDR  DATA  IO  RW  DMA  M1  EXTR
  B0   1       =XXXX   XX    X   X   0    X  XXXX
  B1   0       =XXXX   XX    X   X   0    X  XXXX
  B2   0       =XXXX   XX    X   X   0    X  XXXX
  Q    0       =XXXX   XX    X   X   0    X  XXXX

----------------------------------------------------------------------------------
                          TO CONTINUE TYPE  ^
----------------------------------------------------------------------------------

          B0:  Break immediately on all but DMA cycles
          B1:  Ignored.
          B2:  Ignored.
          Q :  Store all traces except DMA but don't store
                    after breakpoints are satisfied.
          T :  Timer times in bus cycles.


>
**********************************************************************************
**********************************************************************************
**********************************************************************************

                        LOGIC ANALYZER DISPLAY

  AND  COUNT    ADDR  DATA  IO  RW  DMA  M1  EXTR      PC=0000  Eaddr=0000
  B0   1       =XXXX   XX    X   X   0    X  XXXX
  B1   0       =XXXX   XX    X   X   0    X  XXXX      TIMER=16777215 cycles
  B2   0       =XXXX   XX    X   X   0    X  XXXX
  Q    0       =XXXX   XX    X   X   0    X  XXXX

----------------------------------------------------------------------------------
                          TO CONTINUE TYPE  ^
----------------------------------------------------------------------------------

                              PC=XXXX
     XXXX is the contents of the program counter from the Debugger
     display.

                             Eaddr=XXXX
     XXXX is the address of the memory pointer on the active side
     of the Debugger display.


>
**********************************************************************************
**********************************************************************************
```

****
****

LOGIC ANALYZER DISPLAY

Two display formats are available to users of the Logic Analyzer: a parallel state (hexadacimal) display and a serial (timing diagram) display.

TO CONTINUE TYPE ^

The parallel state display is most useful when comparing program execution history stored by the Logic Analyzer with a program listing.

The serial display is most useful in determining the timing relations between signals.

****
****

The next three displays explain the parallel display in the Logic Analyzer.

To continue, enter ^.

****

PARALLEL STATE DISPLAY

FP

This command calls the parallel state display. The parallel state display shows 16 traces at a time in hexadecimal form. Each trace represents the values of 48 lines during one bus cycle. The left half of the screen displays the dissassembled program; the right half displays the bus data.

TO CONTINUE TYPE ^

A reverse video line in the middle of the display shows the position of the trace relative to the trace causing break conditions to be satisfied (or the last trace stored if the break conditions were not satisfied.) The break trace line is highlighted.

>

****

PARALLEL STATE DISPLAY

The BREAK+ or BREAK- indicator of the reverse video line shows the number of qualified bus cycles between the reverse video trace and the trace causing the break conditions to be satisfied. (or the last trace stored.)

TO CONTINUE TYPE ^

<UP ARROW>

This command moves the reverse video line back one line in the Logic Analyzer trace buffer. No action if at the end of the buffer.

>

****

****
****

PARALLEL STATE DISPLAY

<DOWN ARROW>

This command moves the reverse video line down one trace in the Logic Analyzer trace buffer. No action if at end of buffer.

TO CONTINUE TYPE ^

P+{count} or P-{count}

This command moves count traces foward (or backward) in the Logic Analyzer trace buffer. count must be a decimal number between 0 and 255 inclusive.

****
****

After entering ^, the module jumps to the Debugger and goes to the Logic Analyzer display. The command P-20 is displayed.

When <RETURN> is entered, the reverse video line moves back 20 lines in the Logic Analyzer storage buffer. Notice that the reverse video line now displays BREAK-20. The command P+5 is displayed on the command line.

When <RETURN> is entered, the reverse video line moves forward 5 lines in the Logic Analyzer storage buffer. Notice that the reverse video line now displays BREAK-15.

To continue, enter ^.

****

SERIAL (TIMING DIAGRAM) DISPLAY

FS[,line name][,line name]...

This command displays the serial or timing diagram of the contents of the Logic Analyzer trace buffer. When line names are specified, the lines are reordered on the display in that order.

TO CONTINUE TYPE ^

The timing diagram is constructed from the data stored in the Logic Analyzer and uses four special graphic characters to produce the timing diagrams. 64 bits of information, representing the status of one line, are examined and the appropriate symbol is produced for each bit. The status of 16 lines is displayed at one time.

>

****

The next three displays and two pages explain the serial or wave form display in the Logic Analyzer.

To continue, enter ^.

****
****

SERIAL (TIMING DIAGRAM) DISPLAY

There is a reverse video area in the middle of the display allowing the user to compare the timing diagrams on the CRT screen. Above that line the BREAK+ or BREAK- indicator shows the number of qualified bus cycles between the reverse video trace and the trace.

TO CONTINUE TYPE ^

The timing diagram is continuous only if the Logic Analyzer is storing all machine cycles. Thus all trace Qualifier (Q) parameters which affect storage must be set to "Don't Care (X)".

****
****
****

SERIAL (TIMING DIAGRAM) DISPLAY

<RIGHT ARROW>

This command moves the displayed data one trace to the right. This results in displaying the next earlier trace in the timing sequence. No effect if at the end of the trace buffer.

TO CONTINUE TYPE ^

<LEFT ARROW>

This command moves the displayed data one trace to the left. This results in displaying the next later trace in the timing sequence. No effect if at the end of the trace buffer.

****
****

****

SERIAL (TIMING DIAGRAM) DISPLAY

P+{count}

This command is used to quickly view events later in the trace buffer. The display moves count traces to the left. count is a decimal number between 0 and 255. The display will not move past the end of the trace buffer.

TO CONTINUE TYPE ^

P-{count}

This command is used to quickly view events earlier in the trace buffer. The display is moved count traces to the right. count is a decimal number between 0 and 255. The display will not move past the end of the trace buffer.

>

****

After entering ^, the module jumps to the Debugger, goes to the Logic Analyzer and displays the command "FS".

When <RETURN> is entered, the Logic Analyzer display changes from the parallel display to the serial display. The command "FS,A00,DMA,A07" is displayed. Notice that the address lines are displayed.

When <RETURN> is entered, the lines displayed are reordered. Now the top lines are A00, DMA, A07, A15,... The command "FS,IO,M1,DO,EO" is then displayed.

When <RETURN> is entered, the lines displayed are reordered. Now the top lines are IO, M1, DO, EO, A15,... The command "P-40" is then displayed.

When <RETURN> is entered, the display is moved back 40 lines in the buffer. Notice the reverse video line is now a BREAK-40. The command "P+10" is also displayed.

When <RETURN> is entered, the display is moved foward 10 lines in the buffer. Notice the reverse video line is now at BREAK-30.

The user may now use the cursor control keys to change the Logic Analyzer display.

To continue, enter ^.

****
****

EXECUTING A PROGRAM WITH LOGIC ANALYSIS

To begin Logic Analyzer operation, the user must select the Debugger (JD), load his program (L), and go to the Logic Analyzer (A).

TO CONTINUE TYPE ^

After setting the parameters to his particular needs, the user may execute his program either from the Logic Analyzer or the Debugger.

>

****
****

The next three displays explain the effects of starting execution in the Debugger and starting execution in the Logic Analyzer.

To continue, enter ^.

****

EXECUTION FROM THE LOGIC ANALYZER

N{expression} or E{expression}

Execution begins at the value expression which is any allowable combination of addresses or symbolic references. N will execute with maskable interrupts disabled and E will execute with maskable interrupts enabled.

TO CONTINUE TYPE ^

N$ or E$

Execution begins at the current program counter (PC=) with interrupts disabled (N) or interrupts enabled (E).

>

****

EXECUTION FROM THE LOGIC ANALYZER

E

Execution begins at the address of the cursor on the active side of the Debugger display (Eaddr=).

TO CONTINUE TYPE ^

When program execution is begins in the Logic Analyzer display mode, both the storage process and program execution halt when the Logic Analyzer breakpoints are satisfied. The Logic Analyzer display returns to the screen with the message "Break conditions satisfied". If the breakpoints have not been satisfied, the Debugger display returns.

>

****

****
****

EXECUTION FROM DEBUGGER

When program execution begins in the Debugger display mode, the storage process in the Logic Analyzer halts when the Logic Analyzer breakpoints have been satisfied, and the program continues until a Debugger breakpoint is encountered.

EXAMPLE NEXT, TO CONTINUE TYPE ^

The Debugger display returns to the screen when execution is halted. If the Logic Analyzer breakpoints were satisfied, the message "Break conditions satisfied" flashes on the screen. If the Logic Analyzer breakpoints were not satisfied, the Logic Analyzer stops storing data several instructions after program execution has stopped. This is part of the Debugger code and should be ignored.

>

****
****

After entering a ^, the module jumps to the Debugger and displays the command "LINCR,,1000".

When <RETURN> is entered, the file INCR is loaded into memory and the symbol table is loaded into memory starting at 1000H. The PC is 300H, the entry point for the program, and the message "SYMTAB LENGTH=001D" is displayed. The command "D#START" is displayed on the command line.

When <RETURN> is entered, the memory pointer is at location 300H in memory and the command A is displayed.

When <RETURN> is entered, the Logic Analyzer is displayed. The command "BO,C=100,D=FF,A=XX00,IO=1,RW=W" is displayed on the command line.

When <RETURN> is entered, BO is satisfied by the 100th IO write to port X'0'. The command "E#START" is also displayed.

When a <RETURN> is entered, the program is executed and breaks when BO is satisfied. The display returns with the flashing message "Break conditions satisfied".

When ^ is entered, the command BO is Displayed.

When <RETURN> is entered, breakpoint 0 is cleared and the command "B0,C=1,A=#START,M1=1" is displayed.

When <RETURN> is entered, B0 is now satisfied the first time there is an instruction fetch from 300H. The command "B1,C=10,A=#START,M1=1" is displayed.

When <RETURN> is entered, B1 is now satisfied the tenth time there is an instruction fetch from location 300H. The command "TU" is also displayed.

When <RETURN> is entered, the timer now times in microseconds. The command "Q,C=10" is also displayed.

When <RETURN> is entered, the Qualifier register now causes the Logic Analyzer to continue for ten more M cycles after B1 is satisfied. The command D is now displayed.

When <RETURN> is entered, the display returns to the Debugger and the command MR is displayed.

When <RETURN> is entered, the memory and CRT refresh are turned off and the command A is displayed.

When <RETURN> is entered, the display returns to the Logic Analyzer and the command "E#START" is displayed.

When <RETURN> is entered, the program is executed and the Logic Analyzer display returns 10 M cycles after B1 is satisfied. The timer shows the time for executing the loop 10 times.

To continue, enter ^.

When ^ is entered, the command "Q,C=0,A=XX00,IO=1,RW=W" is displayed.

When <RETURN> is entered, the Logic Analyzer now stores only IO writes to port X'0'. The command "E#START" is shown.

When <RETURN> is entered, execution is begins and the Logic Analyzer display returns when B1 is satisfied.

When ^ is entered, the command D is displayed.

When <RETURN> is entered, the Debugger display returns and the command "LCOUNTER,,1000" is displayed.

When <RETURN> is entered, the program in COUNTER is loaded and the command A is displayed.

When <RETURN> is entered, the display returns to the Logic Analyzer, the module clears the breakpoints, and the command "B0,A=COUNT0,M1=1" is displayed.

When <RETURN> is entered, B0 is now satisfied by the first instruction fetch to location 100. The next command, "B1,C=2,A=#DATA,RW=W" is displayed.

When <RETURN> is entered, B1 in now be satisfied by the second write to location 200H. The command "Q,C=10" is also displayed.

When <RETURN> is entered, the Qualifer register now causes the Logic Analyzer to continue storing information for 10 M cycles after B1 is satisfied. The command "E#COUNT" is also displayed.

When <RETURN> is entered, the Logic Analyzer clocks the time needed in micorseconds to do two writes to location 200H. The display returns to BREAK+10.

This is the end of the Logic Analyzer module. The user is now free to use any of the options of the Debugger or the Logic Analyzer.

To return to the menu, enter ^.

GenRad
Development Systems Division

Technical Publications Remarks Form

Please use this form to submit your suggestions for revisions, corrections, or additions to this publication. Your comments will be promptly investigated by appropriate technical personnel, and action will be taken as required. If your answer to any of the questions is 'NO' or requires qualification, please include any additional comments on a separate sheet and enclose it inside this pre-addressed form.

TITLE: _____

ORDER NO. _____

DATED _____

	YES	NO
DOES THIS DOCUMENT MEET YOUR NEEDS?	_____	_____
IS THE MATERIAL CONTAINED IN THIS DOCUMENT:		
ACCURATE?	_____	_____
EASY TO READ AND UNDERSTAND?	_____	_____
ORGANIZED FOR CONVENIENT REFERENCING?	_____	_____
WELL ILLUSTRATED WITH USEFUL EXAMPLES?	_____	_____
COMPLETE?	_____	_____

ERRORS IN THE PUBLICATION

SUGGESTIONS FOR IMPROVEMENT

(Please Print)

FROM: NAME _____ DATE _____

ORGANIZATION _____

ADDRESS _____

TITLE _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 608 CULVER CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

GenRad
Development Systems Division

5730 Buckingham Parkway
Culver City, CA 90230

ATTN MARKETING TECHINCIAL PUBLICATIONS

NO. CAN- 104	TITLE: RDOS II RELEASE	DATE: 11/10/80	PAGE 1 of 9

This application note describes the new release of the GenRad/DSD 2300 Series Development System Operating System, RDOS II. The software contains many new features to help you in your development task.

In order to make these functions available to you as rapidly as possible, shipment of this update has started even though two manuals are not yet ready. They are:

MICROPROCESSOR PERSONALITY MANUAL

Information in this manual related to debugging and in-circuit emulation can be found in the following manuals which were shipped with your system:

System Reference Manual (2300-5000) chapters 11, 12
Emulator Reference Manual (2300-5002)
Logic Analyzer Manual (2300-5004)

PROGRAMMERS REFERENCE MANUAL

Information in this manual is an update to the current manual, Programming the ADS (2300-5001).

The Personality and Programmers manuals will be sent to you when they become available.

Your update package should include the following:

3 diskettes
Software Reference Manual
System Reference Card
Debugger Reference Card
ICOM Overview
Application Note package

The diskettes are:

1. System Disk - This disk contains the complete operating system, in a fully functional form. To use the system, simply insert this disk into Drive "0". The File Manager, Editor, Assembler, Linker and Debug Modules can be accessed by entering the appropriate "J" command. All features are described in the Software Reference Manual and the System Reference Cards.

2. Utilities Disk - This disk contains program modules in both Source and Relocatable form. The Relocatable files are discussed in the Programming Reference Manual, and may be used by user applications wishing access to Development System resources (e.g., CRT Disk, Printer). This disk is also used in the event the Debugger needs to be relinked for relocation or reconfiguration.

3. Debug/Logic Analyzer Disk - This disk contains command files and program modules used to relocate the Debug module in memory. The relocation process is necessary in the event that there is a memory conflict between the Debugger and ROM simulation memory assigned to the Development System.

8080/8085/Z80 SUPPORT SYSTEM DISK

The system disk contains the files needed to run the 2300 Development System. Several features have been added to each operating system function. These features are summarized here, full descriptions can be found in the manual. The disk contains nine files. The "**" in the file name shown in this note are replaced by a processor type on the disk directory.

FILE	FEATURES
ASMR**	The ASSEMBLER,entered by (JA),has not been changed functionally, though the following changes have been made to the command menu. The following user options have been deleted. (M) The ASSEMBLER now always prompts for a macro file name. (O) The ASSEMBLER now always prompts for relocatable output file name. (P) The ASSEMBLER now always prompts for a listing file name. This file is used to store the assembler listing for viewing in the Editor. To print the listing; specify P: instead of a file name. (B) To specify a serial printer use the serial port specification: S1: or S2: in place of a listing file name (above). The following user option has been added. (Z) Flag all op-codes which do not produce valid 8080 object code. The user must change any command file which he has created that uses the assembler to reflect these changes.
BINTEL**	The BINTEL program, entered by (JB), bidirectionally converts GenRad/DSD Binary-Intel hex/ASCII absolute object file formats. It can be used in conjunction with the up/download communications command, ICOM (JI), or to drive Intel compatible PROM programmers. Application information is available in the file BINTEL.DOC on the Utility disk supplied.

8080/8085/Z80 SUPPORT

SYSTEM DISK

FILE	FEATURES
DEBUG**D DEBUG**L	The DEBUG**D, software is now divided into a small and large version. DEBUG**D, the small version, is resident from A300 to E000. It is supplied as a "Z" file and is accessed by the command "JD". DEBUG**L is the large version with both logic analyzer and GenRad system PROM programmer support. It is resident from 6E00 to A000 and from C000 to E000. To use this debugger, redefine DEBUG**L with a "Z" attribute, and attribute Debug**D to a 0 attribute then use "JD". The debugger is both reconfigurable and relocatable. The relocatable modules are on the Logic Analyzer disk (supplied). The process of relocation is described in the Personality Manual.
EDITOR**	The EDITOR,entered by (JE), contains several new features including Block Move and File Merge. They are described in the manual. Four undocumented commands have also been added. D[n] Display the directory of disk [n] on the screen from the EDITOR. The edit workspace is not disturbed. TA Set TABS to the assembler language default at locations 11, 19, and 40. TT Set TABS at every 8th space. CTRL-C Upper/lower case toggle within the EDIT input mode.
ICOM**	The ICOM, entered by (JI), up/download utility allows the GenRad/DSD Development System to communicate to other systems via serial port 1. The user can define protocols, hand shake and baud rates needed for the RS-232 interface. An application note is included. More information can be found in the file ICOM.DOC on the Utilities disk (supplied).

8080/8085/Z80 SUPPORT

SYSTEM DISK

FILE	FEATURES
LINKER**	The LINKER, called by (JL), has not been changed functionally, though the following changes have been made to the command menu. The following user options have been deleted: (O) The LINKER now always prompts for output file name. (P) See ASSEMBLER (P) option. (B) See ASSEMBLER (B) option. Command files must be modified to reflect these changes. --- WARNING THE LINKER WILL ONLY ACCEPT FILES PRODUCED BY THE ASSEMBLER IN THE RDOS II RELEASE. --- Programs assembled under previous releases must be reassembled for use with the RDOS II LINKER.
MANAGER	The MANAGER (JM) has been expanded to accept "wild-card" file designations, display alphabetic directories, initialize the serial ports and other functions. The new features are described in detail in the Software Reference Manual.
^CFP	The COMMAND FILE PROCESSOR has not been changed.

RDOS II UPDATE/RELEASE

8080/8085/Z80 Support

Debug/Logic Analyzer Disk

The Debug/Logic Analyzer Disk provides the user with the information and program modules to relocate the debugger in memory. A detailed procedure to perform relocation is contained in the microprocessor Personality Manual.

FILE	EXPLANATION
DEBUG.DOC	Documentation and command files which automatically relocate the standard Debugger.
DBL**.DOC DBL**.LCG	Documentation and command files which automatically relocate the Debugger with logic analyzer and PROM programmer support.

The remainder of the files on the diskette are the relocatable modules needed for relocation.

RDOS II UPDATE/RELEASE

8080/8085/Z80 Support

Utilities Disk

The Utilities disk contains two types of files: Documentation files, which explain the usage of various operating sytems features, source and relocatable system modules used to allow user written programs to access the development system's I/O disk and keyboard/CRT features.

The following documentation files provide detailed information about operating system features. They contain information supplemental to that contained in the Software Reference Manual.

FILE	EXPLAINATION
BINTEL.DOC	Describes the use of the GenRad Binary/Intel Hex conversion program.
ICOM.DOC	Describes the use of the Intersystem Communications program.
NAMING.DOC	Describes the GenRad file naming conventions.
SERIAL.DOC	Describes how to format data to be passed to a Serial Port under control of a user written program.

The following documentation files provide information about linking unique I/O devices to, and relocating, any operating system program such as the Manager, Editor, etc. These documentation files are provided to show in general how this is accomplished. Before attempting this process, please contact your Factory Applications Engineer. He will supply you with the needed software and manuals.

FILE	EXPLAINATION
DRIVER.DOC	Describes in general how to link new I/O drives to the operating system.
SYSTEM.DOC	Describes the final control block (FCB) structure used in performing system I/O. This information is supplemental to the Programming Reference Manual. (2300-5001).
USAGE.DOC	Describes the process of relocating any operating system function. Other diskettes are needed and available on special request from the factory.

8080/8085/Z80 Support Utilities Disk

The remainder of the files on this disk are source and relocatable object files for the system I/O drivers. Their operation is described in detail in the Programming Reference Manual (2300-5001).

FILE	EXPLANATION
CRT80.R	Relocatable CRT driver.
DEVICE.N DEVICE.R	Source and relocatable files containing device table information used for linking new I/O devices.
DEXEC80.R	The Debug executive used in conjunction with the Debug and Logic Analzyer disk to relocate the Debug system program.
DISK 80.R	Relocatable disk I/O driver.
DSP8001 DSP80.R	Dummy serial port handler used with the relocatable Debug system program.
EXEC80.R	Used in conjunction with DEXEC80.R for serial and parallel port support.
FNX80.r	Relocatable file name and unit number control block.
KB80.R	Relocatable keyboard input routines to perform keyboard input.
KEYIN80.R	Single character input from keyboard or command file.
KIO80.R	Line at a time output to CRT.
LINEIN80.R	Reads a line of characters from keyboard or command file.
LP8002	Source and relocatable files for parallel line printer driver, configured to be Centronix compatible.
MEMSIZ80.R	Relocatable program which determines system memory size.
PIO80.R	Relocatable I/O routines used to format information for serial or parallel ports.
SCAN80.R	Relocatable file name scanner.

8080/8085/Z80 Support

Utilities Disk

FILE	EXPLANATION
SP8001 SP80.R	Source and relocatable serial port driver routines.
SYSMAC.N	A source file containing system macro files.
TABLE.N	Source and relocatable programs which associate device names with logical unit numbers.
WORK.R	Relocatable program module used to pass information between system program.

CUSTOMER APPLICATION NOTE

NO. CAN-106	TITLE: I/O PORT USAGE GUIDELINES	DATE: 11/10/80	PAGE 1 of 1

When I/O is not mapped during emulation or simulation, the following ports might create I/O contention problems when accessed by the user program:

Port Address 8080/85/Z80	Port Address 6800/02	Associated Function
F6*+	FFF4-FFF7*	Emulator Control
F1, F3	D0F1,D0F3	Break Key
F8-FE	D0F8	Debug/PROM Pgmr.
AC-B7	D0AC-D0B7	Logic Analyzer
1F-FF	D01F-D0FF	Memory Protect
F0	D0F0	CRT Page Select
F0-F2	D0F0-D0F2	Keyboard
D8-D9	D0D8-D0D9	Disk
C0-C3	D0C0-D0C3	Serial Port 1
C4-C7	D0C4-D0C7	Serial Port 2
C8	D0C8	Real-Time Clock
CC	D0CC	MPIO Board Control Port
CD	D0CD	Printer Port
CD	D0CD	MPIO Board Status Port
CE	D0CE	Disk Reset
CE	D0CE	Trigger RTC

*These registers alter emulation mode.

+This I/O register is not available during emulation. Attempts to write into port F6 during emulation may produce unpredictable results.

NO. CAN-109	TITLE: DEBUGGING Z80 INTERRUPTS	DATE: 11/10/80	PAGE 1 of 1

The Z80 CPU supports three types of interrupts. Mode 0 is equivalent to the 8080A RST instruction. The interrupt vector is interpreted as a single byte object code, representing the first instruction to be executed following the interrupt acknowledge.

In mode 1, the Z80 automatically assumes the next instruction executed following an interrupt response will be a restart, branching to 38 Hex. In mode 1 no interrupt vector is required.

In mode 2, an 8 bit interrupt response vector from the interrupting controller is combined with the contents of the IV (interrupt vector) register to form a 16 bit address. The contents of the IV register comprise the high order 8 bits and the interrupt response vector the low order 8 bits. The contents of that address, and of address + 1 are loaded into the program counter forming the address of the first opcode fetch (start of interrupt routine).

The mode 2 interrupt is unique to the Z80 and is used in conjunction with the RETI (return from interrupt) instruction. An interrupting controller, such as the Z80 serial I/O controller, will keep its interrupt request logic active until the RETI instruction has been executed. At this time the controller sees an EDH followed by a 4DH on the data bus. This alerts the controller that the interrupt has been serviced, and the interrupt request logic can be disabled. The sequence is similar if more than one controller are daisy-chained and interrupts are prioritized.

To perform mode 2 interrupts, both the user clock and I/O (and interrupts) must be enabled, regardless of whether the program is executing out of host or target memory. Obviously if the program is executing from target memory, memory must also be mapped.

CUSTOMER APPLICATION NOTE

NO. CAN- 110	TITLE: SERIAL PORT	DATE: 11/10/80	PAGE 1 of 1

We have discovered a problem when using a printer on a serial port under RDOS II. The parity options presently defined in the I/O Table for this port are incorrect.

Correct table definitions for mode register 1 are:

00 = Disabled (odd)

10 = Disabled

01 = Odd Enabled

11 = Even Enabled

The erroneous table defines the following:

00 = No parity (Default)

10 = Even

11 = Odd

01 = (Unused code)

This problem has been corrected in RDOS 2.1. Customers who received their systems after 10/81 should have RDOS 2.1. If you are currently operating with RDOS 2.0, you should contact GenRad/DSD applications support group for updated software.

NO. CAN-111	TITLE: CREATING/RECONFIGURING	DATE: 11/10/80	PAGE 1 of 5

THE RDOS II MANAGER

Disks Required:

Disk 1: RDOS II System Files Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II Manager UCD Disk

Steps:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1: MAN80.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">" Type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

THE RDOS II CROSS ASSEMBLERS

Disks Required:

Disk 1: RDOS II System Files Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: Cross Assembler nnnn Disk
Disk 4: RDOS II Assembler UCD Disk

Steps:

1. Insert Disk 1 and Disk 4 in Drives 0 and 1 respectively.

2. Type J^1: ASMR. LCF, nnnn where nnnn is the two, three or four character mnemonic of the particular assembler. Press <RETURN>.

3. When the manager prompts with: "M ASMRnnnn.R,1:ASMR.R/C" insert Disk 3 in Drive 0 and then press <RETURN>.

4. When the manager prompts with: "JL" reinsert Disk 1 in Drive 0 and then press <RETURN>.

5. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

6. When the linker prompts with: "Listing File:" ">" enter a Listing-Filename (if desired) and then press <RETURN>.

THE RDOS II MANAGER

7. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

8. When the manager prompts with: "A ASMRnnnn/C/A=0", reinsert Disk 3 in Drive 0 and then press <RETURN>.
   The new assembler is on the Disk in Drive 0. (It is moved from Drive 1 to Drive 0 and is given an attribute of Z.)

9. Place Disk 1 on Drive 0, Disk 3 on Drive 1 and

   A. Attribute Asmnxxxx = 0
   B. Move 1: Asmnxxxx, Asmnxxxx

THE RDOS II LINKER

Disks Required:

Disk 1: RDOS II System Files Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II Linker UCD Disk

Steps:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1: LINK.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">", type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with "JM" reinsert Disk 1 in drive 0 and then press <RETURN>.

THE RDOS II EDITOR

Disk Required:

Disk 1: RDOS II System Files Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II Editor UCD Disk

Steps:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1: EDITOR.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">", type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

THE RDOS II COMMAND FILE PROCESSOR

Disks Required:

Disk 1: RDOS II System Files Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II C.F.P. Ucd Disk

Steps:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1: CFP80.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">", type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

THE RDOS II 8080/8085 DEBUGGER

Disk Required:

Disk 1: RDOS II System Files Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II 8080/8085 Debugger/ L.A. Ucd Disk

Steps:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1: DBG80.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">" type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

THE RDOS II 8080/8085 LOGIC ANALYZER DEBUGGER

Disk Required:

Disk 1: RDOS II System File Disk
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II 8080/8085 Debugger/L.A. UCD Disk

Step:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1:DBL80.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">" type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

THE RDOS II Z80 STANDARD DEBUGGER

Disk Required:

Disk 1: RDOS II System Files Disk (Z80)
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II Z80 Debugger/L.A. UCD Disk

Steps:

1. Insert Disk 1 and Disk 3 in Drive 0 and 1 respectively.

2. Type J^1:DBGZ80.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

THE RDOS II Z80 LOGIC ANALYZER DEBUGGER

Disk Required:

Disk 1: RDOS II System File Disk (Z80)
Disk 2: RDOS II Utility Files Disk
Disk 3: RDOS II 8080/8085 Debugger/L.A. UCD Disk

Step:

1. Insert Disk 1 and Disk 3 in Drives 0 and 1 respectively.

2. Type J^1:DBLZ80.LCF and then press <RETURN>.

3. When the linker prompts with: "L" insert Disk 2 in Drive 0 and then press <RETURN>.

4. When the linker prompts with: "Listing File:" ">" type Listing-Filename (if desired) and then press <RETURN>.

5. When the linker prompts with: "JM" reinsert Disk 1 in Drive 0 and then press <RETURN>.

The GenRad 2300 Advanced Development System (ADS) uses the Non-Maskable Interrupt (NMI) line for system functions. Sharing the NMI line with the target system is accomplished in several different ways, depending on the application.

CASE 1:

The target system has PROM at zero, uses the NMI and at location X'66' there is a jump to the NMI interrupt routine.

Example of PROM:

Address	Contents	Disassembly	
64	00	NOP	
65	00	NOP	
66	C3	JP X'1037'	; users program interrupt routine
67	37		
68	10		
69	00	NOP	
6A	00	NOP	

Sequence of commands to share the NMI:

	Explanation
JD	Jump to the Debugger
D66	Display location X'66'. Note the address of the jump instruction. This address is DEBUG256 + X'0B' for this Debugger.
U0,01	Set the memory map mask so that the first 256 bytes (from 0 to FF) will be mapped into the target system.
MCM	Enable the memory map and enable the target clock. This allows examination of the first 256 bytes of target memory. The clock is mapped in case the emulation clock and the target clock run at different speeds.
WN:PROM,0,FF	Copy the first 256 bytes of memory new disk file called PROM.
M	Clear the emulation modes so that all resources are located in the development system.
U0,0	This clears the memory map mask (optional).
LPROM	Load the file PROM. Now 0-FF in 2300 RAM are a copy of 0-FF in the target system.
D66	Display location X'66'.

Note down the address of the target NMI interrupt routine (in this caseX'1037').

SC3,11,hh — Store a jump to X'D40B'of DEBUG256+X'0B, where 11 refers to the low order byte and hh refers to the high order byte of the jump instruction.

When an NMI occurs, control will be transferred to the Debugger. Next store the address of the user's NMI at DEBUG256.The address of the start of DEBUG256 is found by subtacting X'0B' from the address stored at X'66".

hh11 — Display the start of DEBUG256 .

S37,10 — Store the address of the target system's NMI routine, low order byte first.

WN:PROM.MOD,0,FF,D400,D401. — Save the modified PROM and Debugger in a new disk file called PROM.MOD.

CASE II:

The target system has PROM at zero, uses the NMI and the NMI interrupt routine begins at location X'66'. The general solution is a cut and patch procedure.

Relocate the first three instructions of the beginning of the interrupt routine to an unused portion of memory. Store the jump to DEBUG256 + X'B' at X'66' and store the relocated start address of the interrupt routine, formerly at X'66', at DEBUG256.

Example:

Contents of PROM:

```
63
64
65
66      F5      PUS     PSW
67      D8      IN      A,(X'6')
68      06
69      E6      ANI     X'1'
6A      02
6B      C2      ZJP     NZ,X'346'
6C      46
6D      03
6E
6F
70
```

Assume that the target system does not use memory above X'6000' (i.e all memory is between and 0-5FFF.

Move the beginning of the interrupt routine to an unused portion of memory, store the jump to DEBUG256+X'0B' at X'66' and store the new start address of the interrupt routine at DEBUG256.

Example:

Contents of PROM:

```
63
64
65
66      F5      PUS   PSW
67      D8      IN    A,(X'06')
68      06
69      E6      ANI   X'01'
6A      02
6B      C2      ZJP   NZ,X'0346'
6C      46
6D      03
6E
6F
70
```

Assume that the target system does not use memory above X'6000' (i.e. all memory is between and 0-5FFF.

Commands to share the NMI and set up emulation:

JD	Jump to the Debugger.
D66	Display location X'66'. Note the address of jump instruction. This address is DEBUG256 + X'0B' for this Debugger.
U0,01	Set the memory mask so that the first 256 bytes of memory will be mapped into the target system.
MCM	Enable the memory map and the target clock. This allows examination of the first 256 bytes of the target memory space. The target clock willbe case it is different from the 2300 clock.
WN:PROM,0,FF	Save the first 256 bytes of memory in a new disk file called PROM.
M	Clear the emulation modes.
U0,0	Clear the memory map mask.
LPROM	Load the copy of the user PROM from diskette into 2300 RAM.
D66	Display location X'66'.
SC3,11,hh,	Store a jump to DEBUG256+X'0B', where 11 refers to the high order byte of the jump instruction.

D6000	Display X'6000' which will be the start of unused memory.
SF5,D8,06,E6,02,C3, 6B,00'	Store copy of code which started at X'66' and a jump past the NMI interrupt vector.
hh11	Display the start of DEBUG256. The address of start of DEBUG256 is found by subtracting X'0B' from the address stored at X'66'.
S00,60	Store new start addresses of the target system's interrupt routine.
WN:PROM.MOD,0,FF,6000, 6005,D400,D401	Save the patches in a new file called PROM MOD.

This is an example to show how to write a command file for emulation. No comments are allowed in a command file.

```
JD                      Jump to the Debugger.
LPROM.MOD               Load patches to target system.
U0,FE                   Map all memory except modified first 256 bytes.
U8,00,FF
U1000,FF
U1800,FF
U2000,FF
U2000,FF
U2800,FF
U3000,FF
U3800,FF
U4000,FF
U4800,FF
U5000,FF
U5800,FF
U6000,FF                6000-6100 for scatch area
U6800,FF
U7000,FF
U7800,FF
U8000,FF
U8800,FF
U9000,FF
U9800,FF
UA000,FF
UA800,FF
UW000,FF
UW800,FF
UC000,FF
UC800,FF
UD000,FF
UD800,FF
UE000,FF
UE800,FF
UF000,FF
UF800,FF
X                       write protect 0-FF MER
```